Studies on the Optimization of Query Processing in Deductive Databases(Dissertation_全文)

AUTHOR(S):

Uno, Yushi

Studies on the Optimization of Query Processing in Deductive Databases

Yushi UNO

STUDIES
ON
THE OPTIMIZATION OF QUERY PROCESSING
IN DEDUCTIVE DATABASES

Yushi UNO

Submitted in partial fulfillment of
the requirement for the degree of
DOCTOR OF ENGINEERING
(Applied Systems Science)

KYOTO UNIVERSITY
KYOTO, 606, JAPAN
MARCH, 1995

Preface

A large quantity of data has been overflowing in our daily life nowadays, therefore it is important for us to extract only the necessary data efficiently and utilize them for our own purpose. In order to do so, many pioneers have constructed the 'database' to store or accumulate the similar kind of data from which we are able to obtain necessary information. Such databases have been commercially developed, and have become indispensable for many institutions such as school, hospital, library, company, especially for bank and railway and airline company, whose essential task is to process a large amount of data.

In the light of its the theoretical aspect, the databases can be said to have started in 1970 when E. F. Codd proposed the so-called relational data model and relational calculus, although a few ideas about the subject had been seen before them. The relational data model prevailed immediately because it was strictly and mathematically formalized, based on the theory of relational algebra, and also has theoretical clarity. It is not too much to say that most of the recent database systems in practical use owe their implementations to Codd's theory.

Since relational databases are always increasing in their size and complexity, the common data among several databases or the data inferred from others are considered to be redundant. For this reason, data are not always saved doubly or

explicitly. Such sophisticated data are sometimes called knowledge. This knowledge is implemented in databases as the forms of rules, which are applied to the data as the forms of facts, for inferring new data. This type of databases to support deductive capabilities are called deductive databases. In short, deductive databases consist of rules and facts. Once relational or deductive databases are constructed, the most important issue for their users is to retrieve necessary data accessed by their queries as efficiently as possible. A large number of studies have been made in this field from both theoretical and practical point of view. Especially, the studies on the efficient query processing for deductive databases have been very active because they require special contrivance, which is not necessary for relational databases, to attain efficiency.

Our studies belong to the field of the optimization of query processing in relational and deductive databases, and the main aim of this thesis is to provide a method to estimate the cost for query processing in advance, in order to find a most efficient one among the numerous existing methods of query processing. The cost depends greatly on the probabilistic distributions of the stored data, and our approach pays much attention to this point.

The importance of efficient query processing will never lose its necessity to all kinds of databases in use, because, as we know, the quantity of data is increasing considerably day by day and shows much more variety. The author hopes that the work contained in this thesis contributes to develop efficiency in processing databases, and helps further studies in this significant field.

March, 1995

Yushi Uno

Acknowledgement

The author is heartily grateful to Professor Toshihide Ibaraki of Kyoto University for his enthusiastic guidance, discussion and persistent encouragement. He commented in detail on the whole work in manuscript, pointing out weakness and recommending improvements. Without his considerable support, none of this work could not have completed.

The author would like to express his sincere appreciation to Professor Toshiharu Hasegawa of Kyoto University for his supervision of this work and continuous encouragement.

A great deal of gratitude also goes to Professor Masako Sato of University of Osaka Prefecture, who has always been ready to give the author valuable advice.

The author is also indebted to Professor Masao Fukushima of Nara Institute of Science and Technology and Associate Professor Hiroshi Nagamochi of Kyoto University for a number of helpful suggestions.

Thanks are due to Associate Professor Shigeru Masuyama of Toyohashi University, and to Associate Professor Masamitsu Ohnishi of Tohoku University, and to all his friends and colleagues in Professor Ibaraki's laboratory for many enlightening discussions on the area of this work.

Finally, but not least, the author would like to express his highest gratitude to his family for their heartful cooperation and encouragement which go beyond the merely academic.

Contents

List of Figures

List of Tables

List of Programs

Chapter 1

Introduction

1.1 Historical Background on the Study of Databases

The studies on databases are said to have its historical origin in 1955 when W. C. McGee developed general programs called Report Generator for data processing [McG 59]. The programs included general routines for sorting data, file management and report generation. Since his proposals, the studies in databases and database systems have been enforced by the inventions of data models, that is, network, hierarchical and relational data models. For example, two methods for modeling the real world data were proposed, network data model and hierarchical data model in 1960's, and a number of database systems based on those data models are made into practical uses. The first commercial database system, which integrated the function of accumulating and retrieving data, was IDS (Integrated Data Store) released by General Electric Co. in 1963. Afterwards, the DBTG (Data Base Task Group) of CODASYL (Conference in Data System Languages) improved the data model implemented in IDS as the DBTG network model, and it is now recognized to be a typical system of network data model. In 1968, IBM released IMS (Information Management System), which was a database management system that employs a hierarchical data model.

Although a few basic ideas of database systems owe their origin to the network and hierarchical data models and were already announced in 1960's, it was in 1970's that the studies on databases came out on the full scale. The trigger was the proposal of a new data model, called a relational data model, by E. F. Codd in 1970 [Cod 70]. This model was entirely different from the former two models in that it was thoroughly and strictly defined by the mathematical theory based on the relational algebra. In other words, it is based on the notion that a database is a set of relations. He insists that if the relational data model is implemented on computers, the data independency, which is the independence between the data and the data manipulation language, would be attained.

Because of its theoretical clarity, the relational data model soon took the position of the network and hierarchical data models. In 1980's, the relational data model was studied and developed into two directions; one is for the study of basic theory of relational data model and its formalization [Cod 70, Gard 89, Ull 89b], and the other is for the development of practical database management systems. Among such practical systems, some of the famous ones are, INGRES by Berkeley of the University of California, and IBM developed System R with the relational database language SEQUEL, which was later standardized into SQL.

Recently, new conceptual models of data, such as object-oriented data model and semantic data model have been proposed from distinct viewpoints [Ull 89b, Ull 90]. The characteristics of such types of new models are said to be superior in the ability of data manipulation, but are inferior in the freedom of data processing because the definition of their data becomes more complex. Whatever new kinds of conceptual models are proposed, they usually utilize the ability of relational data model when those models are further modeled into logical databases in order to implement them on computers.

Thus the importance of the relational data model has never been lost since it was introduced by Codd in 1970.

1.2 Optimization of Query Processing in Database Systems

We observe the importance of optimization of query processing in database systems [Gard 89, Ull 89b] in this section.

In general, the relational data model must provide the following two functions:

- data definition,
- data manipulation language (DML).

In this thesis, we adopt the following two popular models as DML:

- relational algebra,
- datalog.

In the classical databases, data consist only of raw data (facts), and these data are called relational databases. On the other hand, in the recent databases especially written in datalog, they can also store another kind of data called rules as well as facts. They are not raw data but are used for inferring new data. In other words, the database contains 'knowledge" in itself. The databases containing both facts and rules are referred to as deductive databases [Ban 86a, Bay 85b, Gard 89, Kif 86, Miy 89].

Once a relational or deductive database system is constructed, users issue requests for retrieving the database, which are written in the DML provided by the database management system [Ull 85b]. We call this request a (database) query, and the query processing is to derive answers to the query from the databases. In the form of datalog description, we have to consider both non-recursive and recursive queries [Ban 86a, Bay 85b, Gün 86], which require finite and infinite times of retrievals of facts, respectively.

According to the original proposal of Codd, database queries in the relational data model are written by a set of operations defined in relational algebra [Gard 89, Ull 85a, Ull 89b, Uno 92], which consists of some kinds of set operations and some newly defined and unique operations for manipulating single or multiple relations. In order to express

recursive queries, we have to introduce a new operation, called transitive closure [Agr 87, Aho 74, Ioa 86, Ioa 88, Jak 91, Jak 92, Lu 87], which includes the execution of infinite times of join operations, and some generalization of transitive closures [Sip 88].

The query processing is an important task of a database management system (DBMS). First of all, the DBMS has to provide us with a convenient means to express queries, being an good interface between data and their users. In addition to this, a significant issue of practical importance is to derive the answers from data as efficiently (cheaply) as possible.

The way of deriving the answers to a given query from data is not always unique, and there can be alternative methods (strategies) to retrieve and process the raw data. Among these methods, finding the method to attain the highest efficiency among possible methods and process the queries by the selected method is referred to as the query optimization or optimization of query processing. In order to evaluate and compare the efficiency among many methods, the efficiency of each method must be first measured. As a measure of efficiency, we often adopt the time complexity (or the time consumed) for processing the query. We usually use the term "cost" of a method, to imply the time required for obtaining the answers to the query by the method. In other words, the query optimization can be understood as the reduction of the cost for query processing.

There are two main approaches for query optimization:

1. Considering that a query is expressed by a sequence of operations in relational algebra, find a method to make a better expression, which is irrelevant to the characteristics of data (e.g. the distribution of data in a relation), to obtain the same answer, that is, the query transformation [Ban 86a], and
2. Noticing that even if the same method is applied to derive the answer, the cost for query processing depends on the features of data such as the data size or the distribution of data, find a means to measure its cost before actually processing the queries [Haa 92, Kri 86, Lip 89, Lip 90, Lyn 88, Uno 92, Uno 94a]. We then compare the costs of all possible methods in advance, and carry out the actual query processing by using the most economical method.

In the first approach, we often utilize the constants included in the given queries. The so-called magic set method, magic template method and the counting method are the typical among these query transformation techniques [Ban 86b, Bee 87, Cer 87, Gra 93, Gup 92, Han 93, Hen 84, Sek 89, Ull 89a, You 92, Zan 86]. There are other kinds of query transformation, such as those which try to reduce the number of variables involved in the query.

In the second approach, we suppose that the method of query processing is expressed by a sequence of operations in relational algebra and transitive closures. The first important issue here is how to execute each operation efficiently. It is known that the cost of a join operation is computationally rather expensive than the cost of other operations in relational algebra [Gra 93, Kim 80]. This also implies that the cost of transitive closures is also expensive because transitive closure includes computation of many joins. For this reason, studies on the efficient computation of joins and transitive closures have been very active.

Even if the set of applied operations is the same, there are many different sequences of operations, which give the same result (answer). The cost of processing queries critically depends on the sequence of executing these operations. Then, it is crucial to find the sequence that minimizes the resulting cost. Therefore, the studies on scheduling the sequence of operations is one of the main streams [Ibara 84, Kri 86, Ull 89b].

As explained in this section, there are many approaches to achieve efficient query processing. A great deal of effort, having been and being devoted to this field, proves the importance of optimization of query processing.

1.3 Research Objectives and Outline of the Thesis

One of the main aims of this thesis is to provide quantitative estimations of the cost for query processing, which we define as the sum of the costs of the operations of relational algebra used to process the query. Furthermore, we would like to clarify the computational complexity of the problem of optimizing the query processing cost.

Chapter 2 introduces the basic notions of relational database as a data model and prepare several definitions and notations, which are necessary for the study of relational and deductive databases [Gard 89, Ull 89b]. Especially, description of the datalog and the definition of the operations of relational algebra, as well as transitive closures, are provided.

In Chapter 3, firstly we introduce a method of obtaining an approximation estimation of the costs (sizes) of all operations in relational algebra and the operation of transitive closure, assuming that the original relations are randomly generated from the uniform probabilistic distribution. In order to prepare for subsequent discussions, we define the generalized transitive closure, which is defined over two or more attributes and relations. Concerning with the results, we show some computational experiments of the cost of joins and transitive closures, which are considered to be the most important operations. Finally, we present some examples of applying our approximations to practical database queries to show their effectiveness [Uno 92].

Chapter 4 proposes another approximation method to estimate the costs of all operations as an improvement of the method in Chapter 3, under the assumption that the original relations are randomly generated from the general probabilistic distribution (including uniform distribution as a special case). This approximation is rather complicated if compared with the one proposed in Chapter 3, but it is a more accurate approximation even for the case of uniform distribution. It also gives a means to estimate join selectivities (which tell and indicate us the size of the relation after joining two relations)

between any of the arbitrary attributes of relations [Uno 94a]. (The precise definition is given in Chapter 2.)

In Chapter 5 gives a method of computing the exact size of a transitive closure, based on the theory of random graphs and random digraphs, and present an algorithm for computing it by dynamic programming. Then, we present lower and upper bounds on the size of a transitive closure. These bounds can be computed in linear time with respect to its domain (input) size. Throughout Chapters 4 and 5, we also give some numerical experiments to illustrate the effectiveness of our proposals [Uno 94b].

In Chapter 6, we formalize the problem OPTJOIN, which determines the optimum join order when many join operations have to be executed. The join operations under consideration are conventionally represented by query graphs. These query graphs are given with relation sizes and join selectivities. Unfortunately, even if the query graphs are restricted to trees, the problem OPTJOIN turns out to be NP-hard. Then, we find some special classes of query trees, for which there are polynomial time algorithms to solve OPTJOIN [Uno 91].

Finally, in Chapter 7, we summarize our study in this thesis and state the contribution of our study to this field.

Importance of efficient processing of database queries is evident to everybody, as the size of data is always increasing. The author hopes that the work in this thesis will be helpful for making the efficient processing databases possible.

Chapter 2

Preliminaries

2.1 Data, Databases and Data Models

2.1.1 Data

Data is generally recognized and defined as "facts" or "phenomena", based on which computations and inferences are performed. We should notice that data and information are different and should distinguish them. Data are purely raw, and they have no meaning unless we accept them, interpret them and give them some meanings. If a piece of data supplies us with new knowledge that we have not had before, it can be called information.

There exist various kinds of data in our society; personal data, addresses and telephone numbers, school records of students, visual data from television, literature data in the libraries, and so on. No matter what kind of data we are concerned, it is necessary that we should collect them and exploit them efficiently according to our needs.

2.1.2 Database and Query

A *database* is a (logical) model of the data in the real world constructed on a computer. In other words, the data in the real world are modeled by some means in order to im-

plement a database on a computer. Mathematically, a database is a set of data collected systematically. Once a database is constructed, the users of it can make requests for retrieving some useful information from the data in the database. Those requests are called *database query* (or simply *query*), and the set of data that satisfies our requests is referred to as *answer* to the query.

A *database management system* (*DBMS*) is an interface between database and users, i.e., it gives us a means to access a large amount of database easily and efficiently. Roughly speaking, a DBMS supports the ability of modeling the data in the real world and supplies a language for answering queries (often called a query language or data manipulation language (DML)) that allows us to access, retrieve and manipulate the relevant data.

A *database system* consists of two parts; a database and a DBMS.

2.1.3 Data Modeling

As mentioned above, we have to map the data in the real world onto computers in order to implement database on computers. This process is called data modeling.

The process of data modeling consists of the following two phases as shown in Figure 2.1, that is,

1. Transformation of the real world data into conceptual models, and
2. Transformation of conceptual models into logical models (i.e. databases).

A conceptual model is a description of how a database designer recognizes the real world conceptually, and it is introduced as an intermediate model between the data and the databases. The term 'conceptual model' is sometimes used as the means or the formalization of the transformation of the real world data into conceptual models. It does not matter whether a conceptual model can directly be implemented on computers or not.

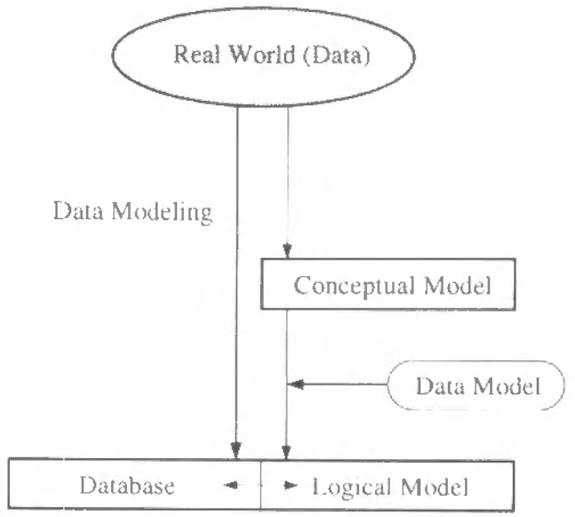


Figure 2.1: Data modeling.

We now describe some of the models proposed to be used in such conceptual models.

- An Entity-Relational (E-R) model

  This is proposed by P. P. Chen in 1976. This model tries to recognize the real world by the concept of entity and relationship. A relationship expresses the semantic relationship among multiple entities. Then, we construct an entity-relationship diagram which connects entities through relationships.

- An Object-Oriented model

  This model describes the real world as directly as we see by the object identities and class hierarchies. This concept is adopted by some object-oriented programming languages, such as Smalltalk-80, C++, and so on.

A data model is a mathematical formalism of transforming a conceptual model into a logical model (this is indeed referred as "database"), which consists of:

1. A definition or a notation for describing data, and

2. A set of operations used to manipulate data (called data manipulate language (DML) or query language, as explained before).

Several proposals for the data models are categorized into three typical models.

- Network data model

  The network data model is a special case of the E-R model, in which relationships are restricted to be binary. Data are connected by pointers, therefore, the entire set of data constitutes a simple directed graph (network). This model is adopted by IDS, which is the first commercial DBMS.

- Hierarchical data model

  In this model, parent data and child data are connected by pointers to construct a tree, and this model includes a number of such trees. As a result, it constitutes a forest. This model was implemented in IMS, which was the DBMS commercialized by IBM in 1968.

- Relational data model

  This model is proposed by E. F. Codd in 1970. Because of its mathematical formality, the relational data model currently becomes the main stream of the data models and implemented in many general DBMSs. Our discussion in this theses is mainly based on this model. For this reason, this will be explained in the next subsection more carefully.

2.2 Relational Data Model

2.2.1 Structure of the Relational Data Model

Among several types of data model described in the previous section, we concentrate on the relational model throughout this thesis. As mentioned in Section 2.1, a data model consists of two parts; the data definition part and the part of operations for data

manipulation.

The relational data model adopts only relations for defining data.

There are two kinds of formalizations as DML, that is,

1. Relational algebra,
2. Relational calculus,

and it is known that the expressive power (of queries) of these two formalizations are equivalent.

The relational algebra is based on the set theory in mathematics and proposed by Codd at the same time of the proposal of relational data model in 1970 [Cod 70]. In the relational algebra, there are 8 operations, and the result of applying those operations to relations as sets becomes the answer to the query.

On the other hand, the relational calculus is based on the theory of (first-order) predicate calculus [Cha 73] and this is also proposed by Codd in 1972. In this formalization, a query is expressed by predicates and the answer is the set of data that satisfies the predicate. Among the relational calculus, what we call the datalog model [Ull 89b] is often used for its notational convenience. It is also known that datalog has at least the equivalent expressive power to the relational algebra.

In the subsequent subsections, we give more details of the relational data model.

2.2.2 Data in the Relational Model

In this thesis, we take *relational algebra* as the mathematical formalism for manipulating data (i.e., DML) in the relational database model, namely, the system of operations in the relational model.

In the relational model, a relation as a set of data is defined as follows.

Let a domain D_i be a finite set of different *values* or *elements* e_{ij}. The number of different values in D_i is called the *size* of a domain D_i, denoted by $|D_i|$ or d_i. The

Cartesian product (or *product*) of domains $D_1, \ldots, D_k$, written $D_1 \times \cdots \times D_k$, is the set of all k-tuples $(c_1, \ldots, c_k)$ such that $c_1 \in D_1, \ldots, c_k \in D_k$, where k is a positive integer. A *relation* R is any subset of the Cartesian product of k domains, and the members of a relation are called *tuples*. A relation which is a subset of a Cartesian product $D_1 \times \cdots \times D_k$, is said to have *arity* k, or simply a k-ary relation. Especially, when $k = 2$, we often call it a *binary* relation. A tuple $t = (c_1, \ldots, c_k)$, which has k *components* $c_1, \ldots, c_k$, is called a k-tuple. The *size* of a relation R is the number of different tuples in R, and we denote it by $|R|$ or r, similar to the notation of the size of a domain.

We usually see a relation as a table, which we call a *relational table*. In doing so, each row of the table represents a tuple, and each column which corresponds to a domain D_i is often given a name, for example A_i, and the name is called an *attribute*. We often use the notation $R(A_1, \ldots, A_k)$ when a relation R has attributes $A_1, \ldots, A_k$. Furthermore, if an attribute A_i of a relation R and a tuple t in R are to be specified, $R.A_i$ and $t.A_i$ are respectively used to denote them.

Example 2.1: We give an example of a relation and a relational table here. Let domains

$D_1 = \{\text{Engineering, Information, Economics}\}$,

$D_2 = \{\text{Math, Database}\}$,

$D_3 = \{1, 2, 3, 4\}$.

Then, a relation *LECTURES* is a subset of the product $D_1 \times D_2 \times D_3$ such as

$$LECTURES = \{(\text{Engineering, Math, 1}), (\text{Information, Math, 1}),$$
$$(\text{Information, Database, 3}), (\text{Economics, Math, 2})\}$$

Here, *LECTURES* is a 3-ary relation and each of the members of *LECTURES* is a (3-) tuple. Now, we show the relational table of *LECTURES* in Figure 2.2. In this relational table, *DEPARTMENT*, *SUBJECT* and *GRADE* are the attributes for domains D_1, D_2 and D_3, respectively. ■

DEPARTMENT	*SUBJECT*	*GRADE*
Engineering	Math	1
Information	Math	1
Information	Database	3
Economics	Math	2

Figure 2.2: A relation as a relational table.

2.2.3 Operations in the Relational Model

There are 8 operations [Gard 89, Ull 85a, Ull 89b] in the relational algebra. These operations are introduced and defined in this subsection.

The operations in relational algebra consist of two groups: one includes the set operations, while the other includes some operations unique to relational algebra. Each group has 4 operations.

- Set Operations

  ⋆ **Union**

  The union of relations R_1 and R_2, denoted $R_1 \cup R_2$, is the set of tuples that are in R_1 or R_2. We apply the union operation to the relations with the same arity, and the corresponding attributes have the same domain. Therefore, all tuples in the result have the same number of components. Even if the attributes of the corresponding domains are different, we can take the union by renaming them into appropriate attributes. This also applies to the operation of set difference and the intersection described below.

  ⋆ **Set difference**

  The difference of relations R_1 and R_2, denoted $R_1 - R_2$ or $R_1 \backslash R_2$, is the set of tuples in R_1 but not in R_2. Relations R_1 and R_2 must have the same arity.

⋆ **Intersection**

The intersection of relations R_1 and R_2, denoted $R_1 \cap R_2$, is the set of tuples that belongs to both R_1 and R_2. Relations R_1 and R_2 must have the same arity. Note that

$$R_1 \cap R_2 = R_1 - (R_1 - R_2)$$
$$= R_2 - (R_2 - R_1).$$

⋆ **Cartesian product**

Let relations R_1 and R_2 have arities k_1 and k_2, respectively. The Cartesian product (product), denoted $R_1 \times R_2$, is the set of all $(k_1 + k_2)$ tuples whose first k_1 components is a tuple in R_1 and the last k_2 components is a tuple in R_2, that is, all the concatenations of a tuple from R_1 and a tuple from R_2.

- Operations in Relational Algebra

⋆ **Projection**

Let a relation R have arity k, and let $A = \{A_{i_1}, \ldots, A_{i_m}\}$ be a given set of attributes selected from the set of all attributes $\{A_1, \ldots, A_k\}$ of R. Then the projection $\pi_A R$, which is also a relation, has set of attributes A. The tuples in $\pi_A R$ consist of the tuples of m components, obtained from the tuples of R by removing all components without corresponding to the attributes in A. Note that, duplicate tuples may be generated in this process, but such tuples are eliminated except only one tuple, because a relation is a set of tuples and does not allow to have duplicate tuples.

⋆ **Selection**

Let a relation R have arity k. The selection $\sigma_C R$ selects all the tuples from R, which satisfy the given condition C. The condition is described in the form of

$$C = \{A_{i_1}\theta_1 e_1, \ldots, A_{i_m}\theta_m e_m\},$$

where θ_i can be a binary arithmetic comparison operator chosen from

$$<, =, >, \leq, \neq \text{ and } \geq .$$

⋆ **θ-join, equijoin and natural join**

The θ-join is a very important operation in relational data model, because it can directly define a relationship between the data in different relations. The θ-join of relations R_1 and R_2 on attribute A_i in R_1 and A_j in R_2 is denoted by $R_1 \underset{A_i\theta A_j}{\bowtie} R_2$, where θ is an arithmetic comparison operator. It is defined as

$$R_1 \underset{A_i\theta A_j}{\bowtie} R_2 = \sigma_C(R_1 \times R_2),$$

where $C = \{A_i\theta A_j\}$. In other words, the θ-join of R_1 and R_2 consists of the tuples in $R_1 \times R_2$ so that the i-th component of R_1 and the j-th component of R_2 satisfy a relation θ. If θ is $=$, which we often encounter in real applications, the θ-join is called the equijoin.

Let $A_{k_1}, \ldots, A_{k_m}$ be all the attributes appearing both in R_1 and R_2. Then the natural join of R_1 and R_2, denoted $R_1 \bowtie R_2$, is a relation defined as

$$R_1 \bowtie R_2 = \pi_A \sigma_C(R_1 \times R_2),$$

where $C = \{R_1.A_{k_1} = R_2.A_{k_1}, \ldots, R_1.A_{k_m} = R_2.A_{k_m}\}$ and A is the set of all the attributes in R_1 and R_2 in the appeared order except $A_{k_1}, \ldots, A_{k_m}$ in R_2, that is, the resulting relation of the join is k attributes less than that of the equijoin.

We now define the general case of the natural join. Even if two relations do not have the same attributes, the natural join can join them by the corresponding attributes that have same domains, and furthermore it does not necessarily join by all the attributes that have same domains or by all the same attributes. This natural join can be defined by specifying a set of attributes $A_{i_1}, \ldots, A_{i_m}$ in R_1 and the same number of attributes $A_{j_1}, \ldots, A_{j_m}$ in R_2, and by considering the condition $C = \{R_1.A_{i_1} = R_2.A_{j_1}, \ldots, R_1.A_{i_m} = R_2.A_{j_m}\}$.

(The domain of the corresponding attributes A_{i_k} and A_{j_k} must be the same.) We can rename the attributes A_{i_k} and A_{j_k} used for the natural join by appropriate new attributes (usually by one of the original attributes A_{i_k} and A_{j_k} of R_1 and R_2). To denote this kind of natural join, we use

$$R_1 \underset{R_1.A_{i_1}=R_2.A_{j_1},\ldots,R_1.A_{i_m}=R_2.A_{j_m}}{\bowtie} R_2,$$

or simply

$$R_1 \underset{A_{i_1}=A_{j_1},\ldots,A_{i_m}=A_{j_m}}{\bowtie} R_2.$$

We distinguish between the latter case and the former case of natural join by specifying the condition under the join symbol $\bowtie$ or not. But we often omit this specification of general natural join in our discussion when the condition is obvious.

Throughout this thesis, we usually use the term "join" to represent the natural join of this type.

⋆ **Quotient**

This operation is not so important for our discussion in this thesis. Moreover, this operation is redundant as we state below, so we omit its explanation here.

We have introduced 8 operations in relational algebra and defined 7 of them. Among them, {union, set difference, product, projection, selection} is called the set of five basic operations, because the rest operations are redundant in the sense that they can be expressed by the combinations of these five basic operations, as seen in the above description.

2.2.4 Datalog on Relational Calculus

Now, we define datalog step by step.

An *atomic formula*, is a predicate symbol with a list of arguments. We use lower case letters for predicate symbols. An argument can be either a variable or a constant. We

use, for convention, lower case letters for constants and upper case letters for variables. We can also use arithmetic comparison predicates (operator) as predicate symbols in datalog. Then atomic formulas, for example `p(X,Y)`, where `p` is a predicate, mean a relation. In this form, we can identify the variables X and Y with the attributes in the relation. Without noticing in particular, we often use upper case letters, such as P, for relations corresponding to the predicate symbols, such as `p`. A *literal* is either an atomic formula or a negated atomic formula, denoted, for example, by $\overline{\mathtt{p}}(\mathtt{a},\mathtt{Y})$. A negated atomic formula is called a negative literal, and otherwise, a positive literal. A *clause* is a conjunctive form of literals.

A Horn clause [Bay 85a, Gard 89, Ull 89b] is a clause with at most one positive literal. Thus, a Horn clause whose predicates are written without arguments for convenience here, is in one of the following forms:

1. A single positive literal, say q,
2. A single positive literal and one or more negative literal

   $$\overline{\mathtt{p}_1} \vee \cdots \vee \overline{\mathtt{p}_n} \vee \mathtt{q},$$

   which is logically equivalent to the form of

   $$\mathtt{p}_1 \wedge \cdots \wedge \mathtt{p}_n \rightarrow \mathtt{q},$$

3. One or more negative literals with no positive literals.

In general, however, the 3rd type of Horn clauses is not usually taken into consideration as datalog expressions because it only implies constraints and does not give any positive information. We follow this convention and consider only 1st and 2nd types in our thesis. We often rewrite the 1st and 2nd formulas in the following Prolog-like style:

1. $\mathtt{q} \leftarrow$,
2. $\mathtt{q} \leftarrow \mathtt{p}_1, \ldots, \mathtt{p}_n$.

A Horn clause of the 1st type (i.e. a predicate p) expresses a relation. Therefore, we refer a Horn clause of the 1st type as a *fact*, and that of the 2nd type as a *rule* [Cha 73, Ull 89b]. A *query* is also regarded as a special kind of a rule and often expressed explicitly in the form such as

$$\texttt{query(X,Y)} \leftarrow \texttt{p(X,Y)},$$
$$\texttt{query(X)} \leftarrow \texttt{p(X,Y)},$$
$$\texttt{query(Y)} \leftarrow \texttt{p(a,Y)},$$

and so on. In a rule, the lefthand side of an arrow is called the *head* and the righthand side the body of a *rule*. Each of the p_i's in the body is called a *subgoal*. A collection of given facts and rules is sometimes called a *logical program*.

Finally, datalog is a logical program, in which no function symbol (except for the 6 arithmetic comparison operators $<, =, >, \leq, \neq$ and $\geq$) and no negated subgoals are allowed.

In order to see the correspondence between relational algebra and datalog expressions, we provide a simple example here.

Example 2.2: Let us consider the following datalog program:

$$\texttt{p(Y)} \leftarrow \texttt{r(X,W)}, \texttt{s(a,W,Z)}, \texttt{t(Z,Y)}, \tag{2.1}$$

where we assume that predicates `r`, `s` and `t` are given as facts or already computed as relations R, S and T. We can decompose and rewrite the datalog formula (2.1) into another form of datalog program such as

$$\begin{cases} \texttt{u(Z)} \leftarrow \texttt{r(X,W)}, \texttt{s(a,W,Z)}, \\ \texttt{p(Y)} \leftarrow \texttt{u(Z)}, \texttt{t(Z,Y)}, \end{cases}$$

by introducing a new intermediate predicate p.

Now, we can give an interpretation of this datalog formula in relational algebra by introducing an intermediate relation U, that is,

$$U(Z) = \pi_{\{Z\}}\Big(R(X,W) \bowtie (\sigma_{\{V=a\}} S(V,W,Z))\Big), \tag{2.2}$$

$$P(Y) = \pi_{\{Y\}}\Big(U(Z) \bowtie T(Z,Y)\Big), \tag{2.3}$$

and so we can obtain a final expression about the predicate p as the following sequence of relational operations

$$P(Y) = \pi_{\{Y\}}\Big(\pi_{\{Z\}}\Big(R(X,W) \bowtie (\sigma_{\{V=a\}} S(V,W,Z))\Big) \bowtie T(Z,Y)\Big), \tag{2.4}$$

from equations (2.2) and (2.3). ■

In a datalog program, predicates that appear only in the body of rules (unless they are defined as facts explicitly), such as `s` and `t` in Example 2.2, can be recognized as facts, and such predicates (relations) are called extensional database (EDB) predicates (relations). On the other hand, predicates that appear in the head of rules, such as `u` and `p` in Example 2.2, are the predicates whose relations are not stored but have to be computed by the EDB relations. These predicates (relations) are called intensional database (IDB) predicates (relations).

In the subsequent discussion, therefore, we use both kinds of expressions, relational algebra and datalog, depending on the context.

2.3 Deductive Database

2.3.1 Recursive Database Query

The definition of datalog permits us to define the recursions in datalog programs.

We can define the recursion by making what we call a *dependency graph* [McK 81, Ull 89b]. Let p_i be predicates. In a dependency graph, vertices are predicates. If the predicate p_j occurs in the body of a rule whose head predicate is p_i, there is an arc from predicate p_j to p_i. A datalog program is *recursive* if its dependency graph has one or more cycles, and otherwise, *non-recursive*. All the predicates that are (not) on cycles are said to be recursive (non-recursive) predicates. A predicate on a cycle of length 1 (loop)

is referred to as (simple) recursive, while a predicate on a cycle of length more that 1 is referred to as mutual recursive. All the rules that (do not) include recursive predicates are called recursive (non-recursive) rules, and finally, a query that gives the answer to the recursive predicate is called a recursive query, and otherwise non-recursive query.

Example 2.3: The following datalog program

$$\begin{cases} \texttt{anc}(\texttt{X},\texttt{Y}) \leftarrow \texttt{par}(\texttt{X},\texttt{Y}), \\ \texttt{anc}(\texttt{X},\texttt{Y}) \leftarrow \texttt{anc}(\texttt{X},\texttt{Z}),\texttt{par}(\texttt{Z},\texttt{Y}), \end{cases}$$

is a typical recursive datalog expressing the ancestor-descendant relation, because its dependency graph in Figure 2.3 contains a cycle (loop). In this datalog, par is an EDB

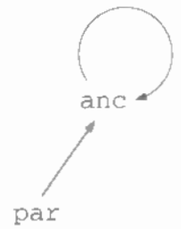


Figure 2.3: A dependency graph.

predicate that means the parent relation, and anc is an IDB predicate that means the ancestor relation. Then, the next query

$$\texttt{query}(\texttt{X},\texttt{Y}) \leftarrow \texttt{anc}(\texttt{X},\texttt{Y}),$$

is a recursive query because it contains a recursive predicate anc in its body. ■

Answering the query and obtaining the anc relation of the recursive query in the example requires to retrieve the par relation recursively. But we cannot translate the recursive rules into the relational algebra expressions, because they require applications of an infinite sequence of operations in relational algebra, such as

$$\begin{aligned} &\texttt{anc}(\texttt{X},\texttt{Y}) \leftarrow \texttt{par}(\texttt{X},\texttt{Y}), \\ &\texttt{anc}(\texttt{X},\texttt{Y}) \leftarrow \texttt{par}(\texttt{X},\texttt{Z}_1),\texttt{par}(\texttt{Z}_1,\texttt{Y}), \\ &\qquad\vdots \end{aligned}$$

It follows from this example that the expressive power of datalog is beyond that of relational algebra. Therefore in the next section, in order to capture the expressive power of the recursive query of datalog, we introduce a new operation, the transitive closure, which includes in itself an infinite sequence of relational operations.

2.3.2 Query Graph

As we have seen, a database query is written by a sequence of relational operations or by datalog. Among relational operations, the join is known as the most expensive and time consuming operation. Concentrating on this fact, a *query graph* [Ibara 84, Kri 86] is introduced to represent how joins are executed in a database query. In a query graph, each node represents a relation and each edge denotes that there exists a join execution between the two relations connected by the edge. The edge is often given a label of the attribute which is used for the join. A *query tree* is a special case of a query graph, in which the shape of the graph is a tree.

In Figure 2.4, we show the query graph (tree) of the database query in Example 2.2. Although this is a non-recursive query, we cannot extend this to recursive queries, because they include infinite number of joins, and therefore, cannot be represented by a finite graph.

2.3.3 Deductive Database

According to the definition of EDB and IDB predicates (relations) in datalog, we also refer to a set of facts as EDB and a set of rules as IDB.

A relational database consists only of EDBs. On the other hand, a deductive database

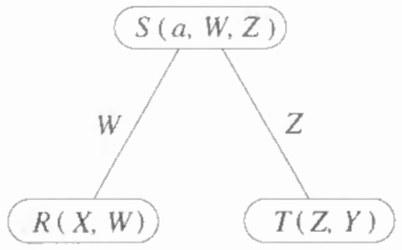


Figure 2.4: A query graph.

consists of both EDBs and IDBs, that is, it can take not only facts but also rules. As a result, by applying rules to facts, a deductive database can produce new facts that are not stored explicitly as EDBs. Since we usually use datalog for expressing queries, a deductive database can be constructed from EDBs and IDBs written in datalog. Thus, a deductive database allows both recursive and non-recursive queries.

When we see the fact that the data can be divided into two forms EDBs (facts) and IDBs (rules) from practical viewpoint, we can say the followings. As the data in relational databases is increasing in size and complexity, common data among several databases or data which is inferred by the other data can be considered to be held as common knowledge not be saved in the form of raw data Those knowledge is implemented as rules in databases, and such databases to support deductive capabilities are called deductive databases. These are the reasons why deductive databases are called the "knowledge base", or are thought to infer knowledge. All of these subjects are studied as a certain field of what we call artificial intelligence [Cha 73].

2.4 Joins and Transitive Closures

2.4.1 Efficiency of Computing Joins

Since the join is the most expensive operation in relational algebra, much effort has been devoted to compute it efficiently [Kim 80, Wol 90]. We explain two typical methods for computing joins, that is, nested-loops and merge-scan algorithms [Gra 93, Ibara 84, Kim 80, Kri 86].

- Nested-loops algorithm

  This is the simplest and most direct algorithm for computing joins. Consider a join $R_1 \underset{A_p=B_q}{\bowtie} R_2$. For the A_p component of each tuple in R_1 (outer loop), this method tries to find all the tuples in R_2 (inner loop) whose values of B_q component coincides with the value of the component in the outer loop, and creates new tuples.

- Merge-scan algorithm

  Consider a join $R_1 \underset{A_p=B_q}{\bowtie} R_2$. We first sort the tuples in R_1 and R_2 by the values of join attributes A_p and B_q, respectively. Then, scanning the tuples in R_1 and R_2 simultaneously in the increasing order of the values of components in B_q for each A_p, we merge the tuples in R_1 and R_2 whose values in the join attributes coincide, into new tuples.

The detail and the implementation as programs of these two methods and some discussions about them will be shown in Chapter 6.

2.4.2 Selectivity of Joins

We have considered the natural join $R_i \underset{A_p=B_q}{\bowtie} R_j$ between two relations R_i and R_j, whose sizes are r_i and r_j, respectively. When considering the size of the resulting relation $R_i \underset{A_p=B_q}{\bowtie} R_j$, it is said to be determined by r_i, r_j and the probability of creating a new

tuple from a pair of tuples randomly chosen from R_i and R_j.

Now, we give the notion of this probability as a definition [Ibara 84, Kri 86, Lyn 88, Ull 90].

Definition 2.1: The *selectivity* s_{ij} with respect to the attribute A_p in R_i and B_q in R_j is the expected fraction of tuple pairs from R_i and R_j that will join among $r_i r_j$ possible pairs, or the probability of creating a new tuple, that is,

$$s_{R_i.A_p,R_j.B_q} = \frac{E\left(|R_i \underset{A_p=B_q}{\bowtie} R_j|\right)}{r_i r_j}, \tag{2.5}$$

where $E(\bullet)$ denotes the expected value of the resulting relation. As it is often the case that the join attributes are obvious from the context, we usually denote the selectivity by

$$s_{ij} = \frac{E\left(|R_i \bowtie R_j|\right)}{r_i r_j}, \tag{2.6}$$

without showing those join attributes A_p and B_q in R_i and R_j. ■

According to this definition, the expected size of the resulting relation of a join $R_i \underset{A_p=B_q}{\bowtie} R_j$ is given by $s_{ij} r_i r_j$. (Alternatively, s_{ij} of the resulting size $s_{ij} r_i r_j$ may be taken as another definition of the selectivity.)

In general, the exact value of the selectivity is difficult to know unless the join is actually carried out. However, it is usually possible to estimate s_{ij} with some accuracy by taking into account the characteristics of the relevant attributes. For example, if a join is computed between attributes A_u of R_1 and B_v of R_2, both with domain D, and if the values of these A_u and B_v are independently chosen with equal probability from D, then we have

$$s_{12} = \sum_{j=1}^{d} \left(\frac{1}{d} \times \frac{1}{d}\right) = \frac{1}{d}. \tag{2.7}$$

More generally, if a value e_j is chosen independently with probability p_{uj} from attribute A_u in R_1 and q_{vj} from attribute B_v in R_2, then we have

$$s_{12} = \sum_{j=1}^{d} p_{uj} q_{vj}. \tag{2.8}$$

For most of the practical cases, a crude but simple estimation (2.7) may be sufficient. In the following discussion, we assume that selectivities are given beforehand by some means or estimated by the assumption that how the relations are generated.

2.4.3 Transitive Closures

As we saw in the previous section, datalog can express recursive queries but relational algebra cannot. In order to provide a means to represent recursive queries by relational algebra, it is necessary to add a new operation, transitive closure [Agr 87, Ioa 88, Jak 91, Lu 87], to the original operations of relational algebra.

There are two alternative approaches to define the transitive closure; one is from the viewpoint of relations, and the other from that of graph theory. Furthermore, there are two kinds of definitions of transitive closures, denoted R^+ and R^*, which we often encounter in practical uses. We see the difference between them in the following definitions.

First, we define a transitive closure in terms of relations.

Definition 2.2: Let R be a binary relation, with two attributes A_1 and A_2 of the same domain D. Then, the transitive closure R^+ and R^* of R is defined as follows:

1. R^+ is the set of all the tuples (e_1, e_2) for which there is a sequence of elements $e_{i_1}, \ldots, e_{i_n}$ $(e_{i_j} \in D)$, such that
   (a) $e_{i_1} = e_1, e_{i_n} = e_2$, and
   (b) $(e_{i_j}, e_{i_{j+1}}) \in R$ for all j $(1 \le j \le n-1)$.
2. R^* is the set of all the tuples (e_1, e_2) for which there is a sequence of elements $e_{i_1}, \ldots, e_{i_n}$ $(e_{i_j} \in D)$, such that
   (a) $e_{i_1} = e_1, e_{i_n} = e_2$, and
   (b) $(e_{i_j}, e_{i_{j+1}}) \in R \cup \{(e_i, e_i)\}$ for all j $(1 \le j \le n-1)$. ■

Now, we give the second definition of a transitive closure in terms of graph theory.

Definition 2.3: Let V be a given set of vertices v_i and A be a set of arcs (v_i, v_j). They define a directed graph $G = (V, A)$. Then,

1. The transitive closure $G^+ = (V, A^+)$ of G has an arc (v_i, v_j) if a vertex v_j is reachable from v_i in G, that is, there is a path of length more than or equal to 1 from v_i to v_j.
2. The transitive closure $G^* = (V, A^*)$ of G has an arc (v_i, v_j) if a vertex v_j is reachable from v_i in G, that is, there is a path of length more than or equal to 0 from v_i to v_j.

The set of arcs A^+ or A^* is also called the transitive closure of G. ■

The definition of the transitive closure in terms of relations can be easily interpreted in the light of graph theory by dint of replacing all the elements e_i in D with all the vertices v_i in G, and all the tuples (e_i, e_j) in R with all the arcs (v_i, v_j) in G. Thus, we see that these two definitions give the same result.

We note here the difference between R^+ (A^+) and R^* (A^*), where R^* permits the reflexive law, or it is equal to R^+ to which all the unit element (e_i, e_i), $e_i \in D$ are added. In terms of graph theory, this means that self-loops (v_i, v_i), $v_i \in V$ are added to G^+. We usually consider the transitive closure R^+, but not R^*, as the transitive closure of a relation R.

When we use the datalog notation, we can give definitions of the transitive closures A^+ and A^* of a graph $G = (V, A)$.

In the following datalog program,

$$\begin{cases} \texttt{path1}(\texttt{V}_1, \texttt{V}_2) \leftarrow \texttt{arc}(\texttt{V}_1, \texttt{V}_2), \\ \texttt{path1}(\texttt{V}_1, \texttt{V}_2) \leftarrow \texttt{arc}(\texttt{V}_1, \texttt{W}), \texttt{path1}(\texttt{W}, \texttt{V}_2), \end{cases} \tag{2.9}$$

where the EDB predicate $\texttt{arc}(\texttt{V}_1, \texttt{V}_2)$ indicates that there is a directed arc from vertex v_1 to v_2, the predicate path1 indicates the transitive closure A^+ of the graph G.

Similarly, in the slight modification of the datalog program (2.9) shown below,

$$\begin{cases} \texttt{path2}(\texttt{V}_1, \texttt{V}_1) \leftarrow, \\ \texttt{path2}(\texttt{V}_1, \texttt{V}_2) \leftarrow \texttt{arc}(\texttt{V}_1, \texttt{V}_2), \\ \texttt{path2}(\texttt{V}_1, \texttt{V}_2) \leftarrow \texttt{arc}(\texttt{V}_1, \texttt{W}), \texttt{path2}(\texttt{W}, \texttt{V}_2), \end{cases} \tag{2.10}$$

the predicate path2 indicates the transitive closure A^* of the graph G.

We often use the notation $\texttt{arc}^+(\texttt{V}_1, \texttt{V}_2)$ ($\texttt{arc}^*(\texttt{V}_1, \texttt{V}_2)$), or simply $\texttt{arc}^+$ ($\texttt{arc}^*$) for expressing the transitive closure path of the predicate arc.

Finally, we define the transitive closure of a single vertex v in a graph $G = (V, A)$.

Definition 2.4: The transitive closure v^+ of a vertex v in a graph $G = (V, A)$ is the answers to the query

$$\texttt{query}(\texttt{X}) \leftarrow \texttt{path1}(\texttt{v}, \texttt{X}),$$

in the datalog program (2.9).

Similarly, the transitive closure v^* of a vertex v in a graph $G = (V, A)$ is the answers to the query

$$\texttt{query}(\texttt{X}) \leftarrow \texttt{path2}(\texttt{v}, \texttt{X}),$$

in the datalog program (2.10). ■

To put it another way, the transitive closure v^+ is the set of vertices that is reachable from v, and v^* is the set of vertices that is reachable from v including itself.

2.4.4 Efficiency of Computing Transitive Closures

Since the cost of joins is rather expensive, compared with other operations of relational algebra, the cost of transitive closures is also expensive because the computation of transitive closures includes many times of computing joins.

Many papers have been published on the efficient computation of transitive closures as well as joins. There can be seen naive algorithm [Ban 86a, Bay 85b, Ioa 88, Lu 87],

semi-naive algorithm [Ban 86a, Bay 85b, Ioa 88, Lu 87] and logarithmic algorithm [Lu 87] among typical methods which are based on computing relations, as well as Warshall's algorithm [Agr 87, Ioa 88, War 62] based on the graph theory. In addition to these classical methods, new approaches are continuously proposed now on [Jak 91, Jak 92]. We give their outline here, and will describe some more details in Chapter 3.

- Warshall's algorithm

  This algorithm computes the transitive closure by manipulating the incident matrix of a graph G, which is the Boolean matrix representation of the vertex-arc incidence relation of G. This algorithm requires the number of iterations that is the same as the length of the longest path in the graph.

- Naive algorithm

  This is the most direct and straightforward algorithm. This algorithm computes $R_1 = R$, $R_2 = R_1 \bowtie R$, $R_3 = R_2 \bowtie R$, ..., successively until no more new tuples are generated.

- Semi-naive algorithm

  The naive algorithm is inefficient because it uses the whole relation R_i generated in each iteration, which includes a lot of duplications of generated tuples. The semi-naive algorithm stores only the new tuples generated at each iteration.

- Logarithmic algorithm

  While the semi-naive algorithm tries to optimize the computation of transitive closures by reducing the intermediate data size involved in the computation, the logarithmic algorithm tries to reduce the number of iterations not up to the length of the longest path in the transitive closure graph, but up to its logarithm. At the 0-th iteration, this algorithm holds the original relation R_1, where R_i includes the pairs of vertices reachable by the path of length i. At the 1st iteration, it computes R_2 by joining R_1 with R_1. At the 2nd iteration, it computes R_3 and R_4 by joining R_2 with R_1 and R_2, and so on. In general, it computes $R_{2^{i-1}+1}$, ..., R_{2^i} at the i-th iteration by joining R_{i-1} with R_1, ..., R_{i-1}. Therefore, this algorithm requires less number of iterations than Naive or Semi-Naive algorithm.

2.5 Optimization of the Cost for Processing Queries

A query to relational or deductive databases can be expressed by datalog and it can also be expressed by operations in relational algebra and transitive closures. To process a query is to find tuples from databases that satisfy the answer predicate according to these operations. The optimization of processing database queries is to reduce the processing cost without changing the answer.

In other words, processing a query is to determine the execution of operations because it is regarded as a finite sequence of operations. The number of orders of operations is numerous enough to examine all of them, although some of them can be known inefficient in advance. But there must be the optimum way of processing among them based on a certain cost measure.

As we mentioned that the process of answering a query is divided into a sequence of applying several operations to relations and that the order of applying those operations is critical to the efficiency of answering a query, we have to define the cost of computing each operation for the purpose of finding the faster and cheaper way of answering a query and optimizing it.

To define the computing cost of each operation, there can be two measures to it;

1. the number of tuples accessed (i.e., scanned, compared, and so on) during the computation of the operation, or
2. the number of tuples outputted as the result of computing the operation.

The measure 1 is usually used for comparing the efficiency between different methods to compute the same operation, such as the nested-loop and the merge-scan methods for

computing joins.

In general, since the processing cost is usually determined by the space complexity of memories and the time complexity for processing, the measure 1 is sometimes not sufficient enough to express the space complexity. Moreover, it is widely believed that outputting the tuples of the resulting relation explicitly is much more time consuming than only accessing tuples. Therefore, the measure 2 is often adopted as the cost of computing each operation [Ban 86a, Kri 86, Ull 90], and we also adopt it in this thesis. This cost not only represents the space complexity but is often dominant in the other complexities. The rightness of this cost in case of join operation will be discussed later in Chapter 6.

The number of tuples outputted as the result of each operation can be said the 'size of intermediate relation' generated after applying the operation. Thus, we can define the processing cost of a query which is a sequence of operations as the sum of the size of intermediate (including the final) relations generated after computing each operation. The importance of this value is obvious from the above discussions. Our main theme in this thesis is to optimize this cost required for processing queries, or to find the optimum order of applying operations.

Chapter 3

Approximate Evaluation of Processing Costs of Uniform Data

3.1 Introduction

Database queries are processed by applying the operations of relational algebra [Ull 89b] or the transitive closure operation. Among such operations, joins and transitive closures are known to be expensive. Therefore, much effort has been devoted to the study of efficient computation of joins and transitive closures [Agr 87, Ioa 88, Lu 87, Sip 88].

The method of obtaining the answer to a database query is not unique, and actually there are many methods of deriving them from databases in general. It is desirable, therefore, to know the most efficient one before we carry it out. In order to do so, we have to evaluate beforehand the computing cost required by each method. Since we can regard that each method consists of a finite sequence of operations in relational algebra and transitive closures, we define the processing cost to be the sum of the cost of these

operations. Therefore, the task is reduced to the evaluation of the cost of each operation.

There can be different viewpoints in defining the computing cost, and in this thesis, we consider it as the size of the resulting relation after applying the operation under consideration [Ban 86a, Kri 86]. This size represents not only the space complexity (as it gives the amount of intermediate data size) but also the other complexities such as the number of computational steps. Therefore, the size can be used as an approximate measure of the computing cost, and the reduction of the intermediate data size directly leads to the reduction of the computing cost.

There are a great deal of results to reduce such computing cost, for some representative form of queries, for example "the same generation query" [Ban 86a, Ban 86b, Bee 87, Gard 89, Miy 89, Sek 89], by transforming them into more tractable forms. Although these results are useful, it is difficult to generalize and apply them to other situations because they are constructed on their unique characteristic of queries, and they cannot give us the computing cost for different data. On the other hand, our approach has generality and convenience. It can be applied whenever we decompose the query processing algorithm into the basic operations.

In this chapter, we evaluate approximately the expected sizes of relations obtained after applying the operations in relational algebra such as selection, projection, natural join, transitive closure of binary relations and its extensions. Although the derivation of approximate formulas is relatively easy for basic operations in relational algebra, it is not so for the operation of transitive closure. For this area, only empirical approaches such as sampling methods were known so far [Haa 92, Lip 89, Lip 90, Lyn 88]. Finally, we use the same generation query as an example of deductive database, and it will be shown that the computing cost for processing this query can be easily evaluated to some accuracy. Based on this evaluation, we can choose the best method in advance among some possible methods of processing the same generation query.

3.2 Computing Cost of Operations in Relational Algebra

In this section, we examine and evaluate the expected size or its approximate value of the resulting relation after the operations of relational algebra are applied. We select, as those operations, Cartesian product, intersection, union, set difference, selection, projection, and natural join, whose definitions are given in Section 2.2, and furthermore, transitive closure in the next section.

3.2.1 Assumptions on Relations

As explained in Chapter 2, a relation does not contain two or more identical tuples in it. To meet this condition, throughout this chapter, relations are considered to be stochastically generated as follows. In a relation R, with domains $D_1, \ldots, D_k$ of sizes $d_1, \ldots, d_k$, there can be $d_1 \cdots d_k$ different tuples. Here, assume that r tuples in R are randomly selected from all the $\binom{d_1 \cdots d_k}{r}$ possible combinations with equal probability $1 \Big/ \binom{d_1 \cdots d_k}{r}$. Therefore, the probability p_{ij} that value (element) $e_j \in D_i$ appears in attribute A_i is $1/d_i$ for all i and j, that is, the values in attribute A_i are generated by the uniform distribution.

Consequently, in the following discussions, we may regard that the elements in any attribute in a relation R are generated stochastically independent from the elements in the other attributes in R. Furthermore, we assume that such an arbitrary relation R_i is generated stochastically independent from any other relation R_j.

While the resulting size of a relation after applying most of the operations to relations is considered to be a stochastic variable, the size r_i of an original relation R_i and the resulting size of Cartesian product $R_i \times R_j$ are not stochastic variables. But we do not distinguish them in their notations in Chapter 3 and 4, for convenience.

3.2.2 Set Operations

Let relations R_1 and R_2 have domains $D_{i_1}, \ldots, D_{i_m}$ and $D_{j_1}, \ldots, D_{j_n}$, respectively. The size of Cartesian product $R_1 \times R_2$ becomes

$$|R_1 \times R_2| = r_1 r_2. \tag{3.1}$$

In the following operations, relations R_1 and R_2 have the same domains $D_1, \ldots, D_k$. Since the probability that an arbitrary tuple in R_1 belongs to R_2 is $r_2/d_1 \cdots d_k$, the expected size of the intersection is

$$|R_1 \cap R_2| = r_1 r_2 / d_1 \cdots d_k. \tag{3.2}$$

By using the above result, we obtain the following formulas about the operations of union and set difference.

$$\begin{aligned} |R_1 \cup R_2| &= |R_1| + |R_2| - |R_1 \cap R_2| \\ &= r_1 + r_2 - r_1 r_2 / d_1 \cdots d_k, \end{aligned} \tag{3.3}$$

$$\begin{aligned} |R_1 - R_2| &= |R_1| - |R_1 \cap R_2| \\ &= r_1 - r_1 r_2 / d_1 \cdots d_k. \end{aligned} \tag{3.4}$$

3.2.3 Selection and Projection

Let the domains of R be $D_1, \ldots, D_k$. Then, let us consider the selection $\sigma_C R$ with a condition C such that

$$C = \{A_{i_1} = e_1, \ldots, A_{i_m} = e_m\}.$$

The expected size of the relation after selection is

$$|\sigma_C R| = r / d_{i_1} \cdots d_{i_m}. \tag{3.5}$$

Let us consider the projection $\pi_A R$ to a given set of attributes $A = \{A_{i_1}, \ldots, A_{i_m}\}$. After taking care of eliminating duplicate tuples, the expected size of the new relation

becomes

$$\begin{aligned} |\pi_A R| &= d_{i_1} \cdots d_{i_m} \left\{ 1 - \left(\frac{d_1 \cdots d_k - d_{i_{m+1}} \cdots d_{i_k}}{d_1 \cdots d_k} \times \right. \right. \\ &\qquad \left. \left. \cdots \times \frac{d_1 \cdots d_k - (r-1) - d_{i_{m+1}} \cdots d_{i_k}}{d_1 \cdots d_k - (r-1)} \right) \right\} \\ &\approx d_{i_1} \cdots d_{i_m} \left\{ 1 - \left(1 - \frac{1}{d_{i_1} \cdots d_{i_m}} \right)^r \right\}. \end{aligned} \tag{3.6}$$

Now, we see the reason why the above formula holds. Consider a tuple, whose elements (values) of $i_1, \ldots, i_m$-th components are specified, (say $c_{i_1} = e_{i_1}, \ldots, c_{i_m} = e_{i_m}$,) and the probability that such a tuple appears in the result of the projection. In the above formula, the product of fractions in the righthand side expresses that such tuples do not appear when we choose r tuples from all the possible $d_1 \cdots d_k$ tuples successively. The bracket { } denotes the probability of the complementary event, that is, a tuple with specified values $e_{i_1}, \ldots, e_{i_m}$ appears at least once in R, and the whole term gives the expected size of different tuples with attributes $A_{i_1}, \ldots, A_{i_m}$.

3.2.4 Natural Join

Suppose that R_1 and R_2 have attributes $A_{i_1}, \ldots, A_{i_m}$ and $A_{j_1}, \ldots, A_{j_n}$, respectively. We consider the natural join $R_1 \bowtie R_2$ which is joined by attributes $A_{i_{s_1}}, \ldots, A_{i_{s_p}}$ in R_1 and $A_{i_{t_1}}, \ldots, A_{i_{t_p}}$ in R_2, where $D_{i_{s_1}} = D_{j_{t_1}}, \ldots, D_{i_{s_p}} = D_{j_{t_p}}$ are assumed. Then, the expected size of the table after the natural join $R_1 \underset{A_{i_{s_1}} = A_{j_{t_1}}, \ldots, A_{i_{s_p}} = A_{j_{t_p}}}{\bowtie} R_2$ is

$$\begin{aligned} |R_1 \bowtie R_2| &= r_1 r_2 / d_{i_{s_1}} \cdots d_{i_{s_p}} \\ &\left(= r_1 r_2 / d_{j_{t_1}} \cdots d_{j_{t_p}} \right). \end{aligned} \tag{3.7}$$

This is because the probability that the values of attributes $A_{i_{s_1}}, \ldots, A_{i_{s_p}}$ of a tuple in R_1 coincide with the values of attributes $A_{j_{t_1}}, \ldots, A_{j_{t_p}}$ of a tuple in R_2 is $1/d_{i_{s_1}} \cdots d_{i_{s_p}}$, and the number of pairs of such tuples is $r_1 r_2$. In other words, the (join) selectivity between R_1 and R_2 (with respect to the attributes $A_{i_{s_1}}, \ldots, A_{i_{s_p}}$ in R_1 and $A_{i_{t_1}}, \ldots, A_{i_{t_p}}$ in R_2) defined in the formula (2.6) in Chapter 2 turns out to be

$$s_{ij} = 1/d_{i_{s_1}} \cdots d_{i_{s_p}}.$$

As a special case, let us consider the natural join of two binary relations. Suppose that R_1 has attributes A_1 and A_2, R_2 has attributes A_3 and A_4, and A_2 and A_3 have the common domain D of size d. Then the resulting size after computing natural join $R_1 \underset{A_2=A_3}{\bowtie} R_2$ is

$$|R_1 \bowtie R_2| = r_1 r_2 / d, \tag{3.8}$$

and the selectivity between R_1 and R_2 is

$$s_{12} = 1/d.$$

3.3 Computing Cost of Transitive Closure

3.3.1 Algorithms for Computing Transitive Closure

Naive Algorithm

In this section, we first explain some basic algorithms for computing the transitive closure R^+ of a binary relation $R(X, Y)$, where the domains of the two attributes X and Y are assumed to be the same domain, which is denoted D.

The first algorithm is what we call Naive algorithm [Ban 86a, Bay 85b, Ioa 88, Lu 87]. This is described in Program 3.1. In line 5 of algorithm Naive1, the attributes X and Y in $R^+(X, Y)$ and $R(X, Y)$ are renamed as $R^+(X, Z)$ and $R(Z, Y)$, respectively. Such an obvious renaming is usually understood from the context, and we sometimes express line 5 as

$$R^+ := R^+ \cup \pi_{\{X,Y\}}(R^+ \bowtie R),$$

by omitting the names of attributes. Conversely, the domains of attributes that are joined must be same, and the corresponding domains of attributes of R^+ in each iteration and the original relation R must also coincide.


```
Procedure Naive1
Input: R
Output: R+
1     begin
2     R+ := R;
3     while 'R+ changes' do
4       begin
5         R+ := R+ ∪ π{X,Y}(R+(X,Z) ⋈ R(Z,Y));
6       end
7     end.
```

Program 3.1: Naive algorithm 1.

In order to represent the related attributes compactly, we also use the notation $\$i$ to indicate the i-th attribute of all the relevant relations. in place of the attribute A_i of R. By using this notation, we write line 5 in the algorithm Naive1 as

$$R^+ := R^+ \cup \pi_{\{\$1,\$4\}}\sigma_{\{\$2=\$3\}}(R^+ \times R)$$

or

$$R^+ := R^+ \cup \pi_{\{\$1,\$3\}}(R^+ \bowtie R).$$

To simplify the notation further, we sometimes denote the composite operation of the natural join and the projection by

$$\pi_{\{X,Y\}}R^+ \bowtie R \equiv R^+ \underline{\bowtie} R.$$

Then, a tuple (e_1, e_2) determined by a sequence $e_1 = e_{i_1}, e_{i_2}, \ldots, e_{i_k} = e_2$ of length k in the transitive closure R^+ is obtained by $(k-1)$ iterations of joining R_1 and R_2. The set of all tuples obtained in this way is denoted by

$$R^k = R \underline{\bowtie} R \underline{\bowtie} \cdots \underline{\bowtie} R \quad (\underline{\bowtie} : k-1 \text{ times}).$$

With this notation, R^+ is expressed by

$$R^+ = \bigcup_{k=1}^{\infty} R^k. \tag{3.9}$$

When we discuss this algorithm as graph theory, it concatenates one or more arcs in the graph G in order to find reachable pairs of vertices until no more new pair is generated. This method is implemented as Warshall's algorithm [Agr 87, Ioa 88, War 62].

Finally, by rewriting the condition 'R^+ changes' into another statement of programs by introducing a new relation ΔR, we can obtain the following algorithm Naive2 of Program 3.2, which is another form of the algorithm Naive1.

```
   Procedure Naive2
   Input: R
   Output: R+
1      begin
2      ΔR := R;
3      Temp_R+ := R;
4      while ΔR ≠ φ do
5        begin
6          R+ :=Temp_R+ ∪ R+ ⋈ R;
7          ΔR := R+−Temp_R+;
8          Temp_R+ := R+;
9        end
10     end.
```

Program 3.2: Naive algorithm 2.

Semi-Naive Algorithm

Now, we describe the Semi-Naive algorithm [Ban 86a, Ban 86b, Ioa 88, Lu 87], which is an improvement of the Naive algorithm. This is based on the idea to store only those tuples newly generated in the i-th iteration, which is denoted by ΔR_i. The algorithm is called Semi-Naive1 and given in Program 3.3. By introducing parameter i for expressing


```
   Procedure Semi-Naive1
   Input: R
   Output: R+
1      begin
2      ΔR := R;
3      R+ := R;
4      while ΔR ≠ φ do
5        begin
6          ΔR := ΔR ⋈ R;
7          ΔR := ΔR − R+;
8          R+ := R+ ∪ ΔR;
9        end
10     end.
```

Program 3.3: Semi-Naive algorithm 1.

the number of iterations for the **while** loop of lines from 4 to 9, we have the algorithm of Program 3.4, which is called Semi-Naive2. In the algorithm of Semi-Naive2, corresponding to (3.9), we compute the transitive closure R^+ as the sum of true increment ΔR_k:

$$R^+ := \bigcup_{k=1}^{\infty} \Delta R^k. \tag{3.10}$$

As the iteration can be stopped whenever $\Delta R^k = \phi$ holds, the number of iterations is finite, and indeed, at most within $(d-1)$ iterations, as we mentioned before. We will evaluate in the subsequent subsections the size of transitive closure from formula (3.10) of the algorithm Semi-Naive2.

```
Procedure Semi-Naive2
Input: R
Output: R+
1     begin
2     ΔR_1 := R;
3     R_1 := R;
4     for i = 2 to d − 1 do
5       begin
6         δR_i := ΔR_{i−1} ⋈ R;
7         ΔR_i := δR_i − R_{i−1};
8         R_i := R_{i−1} ∪ ΔR_i;
9       end
10    end.
```

Program 3.4: Semi-Naive algorithm 2.

3.3.2 Generalization of Transitive Closure

From binary to k-ary relations

The definition of the transitive closure of a binary relation can be easily generalized to the transitive closure of a k-ary relation.

Definition 3.1: Let R be a k-ary relation $R(A_1, \ldots, A_k)$ (the domain of A_i is D_i), and $C = \{A_{s_1} = A_{t_1}, \ldots, A_{s_m} = A_{t_m}\}$. Now, consider the $(i-1)$-th iteration for computing the transitive closure:

$$R_i = R_{i-1} \bowtie R \equiv \pi_{A^{(0)}}\sigma_C(R_{i-1} \times R)$$
$$= \pi_{A^{(0)}}\left(R_{i-1} \underset{A_{s_1}=A_{t_1},\ldots,A_{s_m}=A_{t_m}}{\bowtie} R\right),$$

where $A^{(0)}$ is the set of k attributes which are chosen from the result of $R_{i-1} \bowtie R$, such that their domains are equivalent to those of $A_1, \ldots, A_k$. Furthermore, C is the condition of selection (or natural join) with respect to the m attributes $A_{s_1}, \ldots, A_{s_m}$ in R_{i-1} and $A_{t_1}, \ldots, A_{t_m}$ in R. Then, the transitive closure R^+ by C and $A^{(0)}$ is defined as

$$R^+\left[C; A^{(0)}\right] = \bigcup_{i=1}^{\infty} R_i,$$

where $R_1 = R$ is assumed. ■

From one relation to multiple relations

Next, we consider a more general transitive closure of a relation R which joins R and one or more relations $Q_1, \ldots, Q_u$ at each iteration. After each join operation, projection operation is executed on to the attributes with the same domains as the original relation R. Now, we give a precise definition.

Definition 3.2: Let R be a k-ary relation $R(A_1, \ldots, A_k)$, where the domain of A_i is D_i, Q_h, $h = 1, \ldots, u$ be u relations. Let $A^{(h)}$, $h = 1, \ldots, u$ be sets of attributes belonging to both R and Q_h, where we assume that $A^{(h)}$, $h = 1, \ldots, u$, are disjoint with each other, and let $C = \{R.A^{(1)} = Q_1.A^{(1)}, \ldots, R.A^{(u)} = Q_u.A^{(u)}\}$. Let us consider the following computation as a single and the $(i-1)$-th iteration for computing a transitive closure; the joins of R_{i-1} and u relations $Q_1, \ldots, Q_u$ are performed at the same time as specified by the condition C. The projection of this result onto the set of k attributes $A^{(0)}$, which are chosen from the attributes in R and $Q_1, \ldots, Q_u$, is then performed, and their domains are same as the domains in R. Here, we assume that $A^{(0)} = \{A_1, \ldots, A_k\}$ and $A_1, \ldots, A_v$ are chosen from R and $A_{v+1}, \ldots, A_m$ are chosen from $Q_1, \ldots, Q_u$, without loss of generality. We express the entire computations as

$$R_1 = R,$$
$$R_i \equiv R_{i-1} \bowtie (Q_1, \ldots, Q_u)$$
$$\equiv \pi_{A^{(0)}}\sigma_C(R_{i-1} \times Q_1 \times \cdots \times Q_u)$$
$$= \pi_{A^{(0)}}\left(R_{i-1} \underset{R.A^{(1)}=Q_1.A^{(1)},\ldots,R.A^{(u)}=Q_u.A^{(u)}}{\bowtie} (Q_1, \ldots, Q_u)\right)$$

Finally, the transitive closure R^+ by $Q_1, \ldots, Q_u$, C and $A^{(0)}$ is defined as

$$R^+\left[Q_1,\ldots,Q_u;C;A^{(0)}\right] = \bigcup_{k=1}^{\infty} R_i.$$

■

We note that the transitive closure of a k-ary relations, given in Definition 3.1, is a special case of this definition.

In Definition 3.2, we may consider the sequence of composite operations of joining $Q_1,\ldots,Q_u$, selection by the condition C and projection onto $A^{(0)}$ to be a single operator, which we denote by φ, i.e.,

$$R_i = \varphi(R_{i-1}) = \varphi^{i-1}(R),$$

and therefore, we have

$$R^+ = \bigcup_{i=1}^{\infty} \varphi^{i-1}(R).$$

3.3.3 The Size of Transitive Closure

For the definition of the generalized transitive closures and the operator φ, we can modify algorithm Semi-Naive2, by only replacing line 6 by

$$\delta R_i := \varphi(\Delta R_{i-1}), \tag{3.11}$$

or by introducing the new intermediate relation R'_i,

$$\begin{cases} R'_i := \Delta R_{i-1} \bowtie (Q_1,\ldots,Q_u) \\ \delta R_i := \pi_{A^{(0)}} R'_i. \end{cases} \tag{3.12}$$

In this section, we analyze and evaluate the size of the generalized transitive closure based on the modified version of algorithm Semi-Naive2.

Evaluation of the resulting size by operation φ

First of all, we evaluate the expected size of the relation resulting after operator φ is applied, which is the result of single iteration (3.11).

For this, we first examine the expected size of the relation after natural join $\Delta R_{i-1} \bowtie (Q_1, \ldots, Q_u)$. Let the size of the domain $D_j^{(i)}$ of an attribute $A_j^{(i)} \in A^{(i)}$ used for joining Q_i be $d_j^{(i)}$, and define

$$q_i \equiv |Q_i|, \quad d^{(i)} \equiv \prod_{A_j^{(i)} \in A^{(i)}} d_j^{(i)}.$$

Then, by using formula (3.7) iteratively, we obtain

$$|\Delta R_{i-1} \bowtie (Q_1,\ldots,Q_u)| = \Delta r_{i-1} \times \frac{q_1}{d^{(1)}} \times \cdots \times \frac{q_u}{d^{(u)}} = \Delta r_{i-1} \times p, \tag{3.13}$$

where

$$p \equiv \frac{q_1}{d^{(1)}} \times \cdots \times \frac{q_u}{d^{(u)}} = p_1 q_1 \times \cdots \times p_u q_u, \tag{3.14}$$

$$p_i = 1/d^{(i)}, \quad i = 1,\ldots,u, \tag{3.15}$$

that is, the selectivity s between ΔR_{i-1} and Q_i's is

$$s = \prod_{i=1}^{u} \frac{1}{d^{(i)}}.$$

(There are some cases in which formula (3.15) of the probability p_i of joining the corresponding attributes is not appropriate and given differently, as will be seen in the later examples.)

After the above join operation, we execute the projection to the set of attributes $A^{(0)}$ of the resulting relation. This time, the domain of each attribute $A_j^{(0)} \in A^{(0)}$ is not equal to $D_j^{(0)}$, and its size $d_j'^{(0)}$ should be considered to be

$$d_j'^{(0)} \approx d_j^{(0)} \left\{ 1 - \left(1 - \frac{1}{d_j^{(0)}}\right)^r \right\}, \tag{3.16}$$

because only those values that have appeared in the original R can appear in the new domain. The righthand side of this formula gives the expected value of the number of different tuples when r elements are chosen independently from domain $D_j^{(0)}$ with equal probability. (Here, if the independence of entries is not guaranteed, $d_j^{(0)}$ should be estimated in a different manner.) Then applying formula (3.6) of projection to the

relation $\Delta R_{i-1} \bowtie (Q_1, \ldots, Q_u)$, whose size is $\Delta r_{i-1} \times p$, and to the set of attributes $A^{(0)}$ of size $d_j'^{(0)}$, we have

$$|\varphi(\Delta R_{i-1})| \approx c\left\{1 - \left(1 - \frac{1}{c}\right)^{\Delta r_{i-1} \times p}\right\} \tag{3.17}$$

$$c \equiv \prod_{A_j^{(0)} \in A^{(0)}} d_j'^{(0)}. \tag{3.18}$$

The reason why these formulas are only approximations is that, in addition to that formula (3.6) is an approximation, we use only the expected values of the size of the relation $\Delta R_{i-1} \bowtie (Q_1, \ldots, Q_u)$ and the domain $D_j'^{(0)}$ without considering their distributions, even though they are random variables. In order to emphasize this, we adopt in the following the notations $\delta\rho_i$, $\Delta\rho_i$, ρ_i, ρ^+ instead of δR_i, ΔR_i, R_i, R^+, if they are computed by formula (3.17).

Recursive equation and its solution

As mentioned in Subsection 3.3.2, the transitive closure R^+ of a relation R by operation φ is computed by lines from 6 to 8 in algorithm Semi-Naive2. Corresponding to those relations computed in lines from 6 and 8, we have the following approximate formulas of their sizes,

$$\begin{cases} \delta\rho_i = c\left\{1 - \left(1 - \frac{1}{c}\right)^{\Delta\rho_{i-1} \times p}\right\} & \text{(by (3.17))} \\ \Delta\rho_i = \delta\rho_i\left(1 - \frac{\rho_{i-1}}{c}\right) & \text{(by (3.4))} \\ \rho_i = \rho_{i-1} + \Delta\rho_i. \end{cases} \tag{3.19}$$

Eliminating $\delta\rho_i$ from the first two formulas in (3.19) gives us

$$\begin{cases} \Delta\rho_i = c\left\{1 - \left(1 - \frac{1}{c}\right)^{\Delta\rho_{i-1} \times p}\right\}\left(1 - \frac{\rho_{i-1}}{c}\right) \\ \rho_i = \rho_{i-1} + \Delta\rho_i. \end{cases} \tag{3.20}$$

By formula (3.10), we then obtain

$$\rho^+ = \sum_{i=1}^{\infty} \rho_i. \tag{3.21}$$

However, it is not easy to solve the recursive equation of (3.20) directly. Therefore, taking into account the fact that $1/c \ll 1$ holds in practice, we linearize and approximate (3.20) as

$$\begin{aligned} \Delta\rho_i &\approx c\left\{1 - \left(1 - \frac{1}{c} \times \Delta\rho_{i-1} \times p\right)\right\}\left(1 - \frac{\rho_{i-1}}{c}\right) \\ &= p\left(1 - \frac{\rho_{i-1}}{c}\right)\Delta\rho_{i-1}. \end{aligned} \tag{3.22}$$

Since this approximation formula does not include the second and higher terms of Taylor expansion of $(1 - 1/c)^{\Delta\rho_{i-1} \times p}$, it has a tendency that $\Delta\rho_i$ grows faster than the true value when p grows. Therefore, ρ^+ in (3.21) has the same tendency that it grows faster than the solution of the original recursive equation.

Then we transform formula (3.22) into

$$\begin{aligned} \frac{c}{p} \cdot \Delta\rho_i &\approx (c - \rho_{i-1}) \cdot \Delta\rho_{i-1} \\ &= c \cdot \Delta\rho_{i-1} - \Delta\rho_{i-1} \cdot \rho_{i-1}, \end{aligned}$$

and add these for $i = 2, 3, \ldots$, to obtain

$$\frac{c}{p}(\rho^+ - \Delta\rho_1) \approx c \cdot \rho^+ - \frac{1}{2}\left(\rho^{+2} + \sum_{i=1}^{\infty}\{\Delta\rho_i\}^2\right). \tag{3.23}$$

Here, we replace the last term of the righthand side by

$$\sum_{i=0}^{\infty}\{\Delta\rho_i\}^2 \approx \Delta\rho_3 \cdot \rho^+, \tag{3.24}$$

because it is difficult to compute $\sum_{i=1}^{\infty}\{\Delta\rho_i\}^2$ exactly, and $\Delta\rho_3 \cdot \rho^+$ was observed to be a better approximation than $\Delta\rho_1 \cdot \rho^+$ and $\Delta\rho_2 \cdot \rho^+$ in some numerical experiments, though the lefthand side of (3.24) tends to become larger than the righthand side when p becomes large. Furthermore,

$$\Delta\rho_3 \approx \frac{p^2}{c^3} \cdot \Delta\rho_1 \cdot (c - p \cdot \Delta\rho_1) \cdot (c - \Delta\rho_1)^2, \tag{3.25}$$

as easily shown by applying (3.22) iteratively. As a result, we have

$$p \cdot \rho^{+2} + (p \cdot \Delta\rho_3 + 2c - 2cp)\,\rho^+ - 2c \cdot \Delta\rho_1 \approx 0$$

by (3.23), and

$$\rho^+ \approx \frac{1}{2p}\left\{-(p\cdot\Delta\rho_3+2c-2cp)+\sqrt{(p\cdot\Delta\rho_3+2c-2cp)^2+8cp\cdot\Delta\rho_1}\right\}. \quad (3.26)$$

The approximation (3.26) tends to be larger than the correct value, because the error between (3.22) and (3.24) becomes large for large p. But judging from the later experiments for binary relations, formula (3.26) appears to be sufficient for practical purpose for $p \leq 2$.

Binary relations

We can apply the result of formula (3.26) to a binary relation R. In this case, since the p in formula (3.14) by a single operation $\bowtie$ is given

$$p = r/d,$$

we substitute it into (3.17) and derive

$$\varphi(\Delta R_{i-1}) = |R_{i-1}\bowtie R| \approx c\left\{1-\left(1-\frac{1}{c}\right)^{\Delta r_{i-1}\times r/d}\right\}. \quad (3.27)$$

Formulas (3.25) and (3.26) then become

$$\Delta\rho_3 = \frac{r^3}{d^2}-\frac{2r^4}{cd^2}+\frac{r^5}{c^2d^2}-\frac{r^5}{cd^3}+\frac{2r^6}{c^2d^3}-\frac{r^7}{c^3d^3},$$

$$\rho^+ \approx \frac{1}{2}\left\{-\left(\Delta\rho_3+\frac{2cd}{r}-2c\right)+\sqrt{\left(\Delta\rho_3+\frac{2cd}{r}-2c\right)^2+8cd}\right\}. \quad (3.28)$$

3.3.4 Numerical Experiments

We report the computational results for transitive closures of binary relations R, and evaluate the precision of approximations (3.27) and (3.28) in this section.

Experiment 1: Estimation of $|R\bowtie R|$ by formula (3.27).

For $d = 50, 100$ and 200, and for several values of r for each d, we generate r tuples of R at random, compute $R\bowtie R$ and count its size. Figure 3.1 shows the average size of

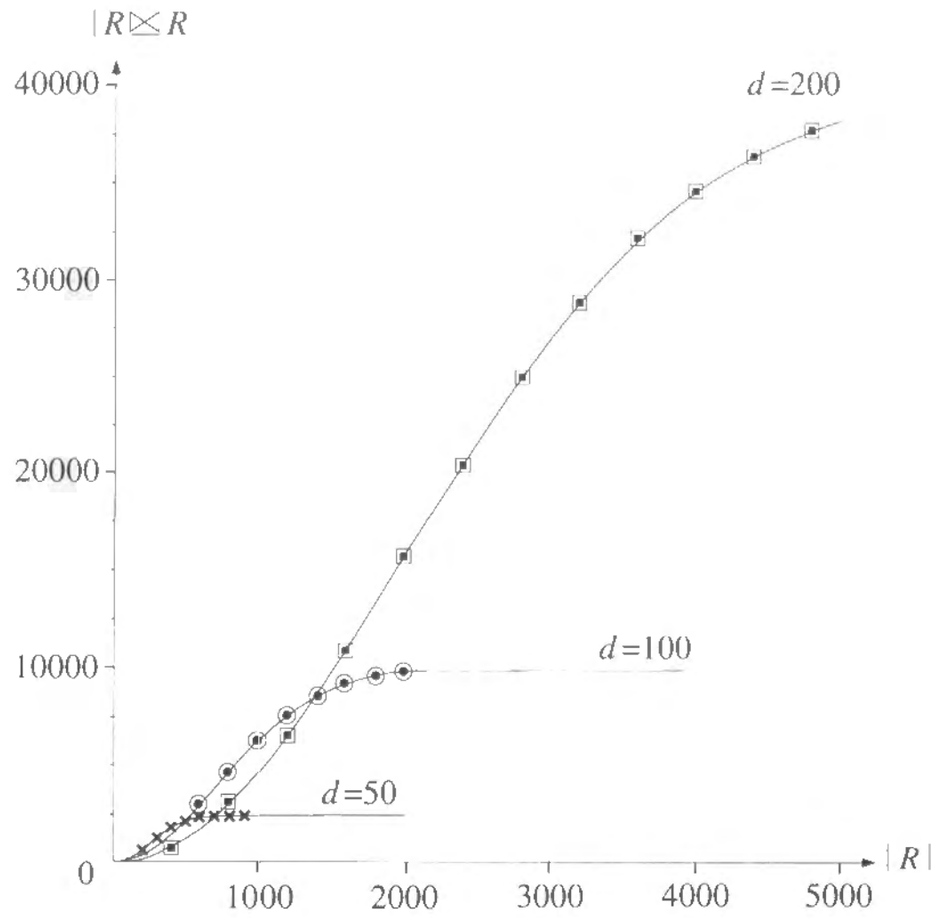


Figure 3.1: Estimation of $|R\bowtie R|$.

$\bowtie$ for 30 trials. In the figure, the solid curves denote the values of formula (3.27) and the dotted circles $\odot$ are the results of our experiments. From these, we may say that formula (3.27) gives a rather accurate approximation.

Experiment 2: Estimation of $|R^+|$ by ρ^+ of formula (3.28).

Figure 3.2 shows the average values of ρ^+ for some selected values of d and r, where the solid curves give the result of solving recursive equation (3.20) until $\Delta\rho_k < 0.01$ is attained, and the cross signs, which are not seen in most of the part in the figure because it mostly coincide with solid curves, give the approximate values given by (3.28), and the dotted circles $\odot$ are the results of our experiments.

From these results, we see that the accuracy of our approximations are satisfactory for the range in which d is rather large, and r is comparatively smaller than d (e.g., in case of $p = r/d \leq 2$), which is usually satisfied in practical applications.

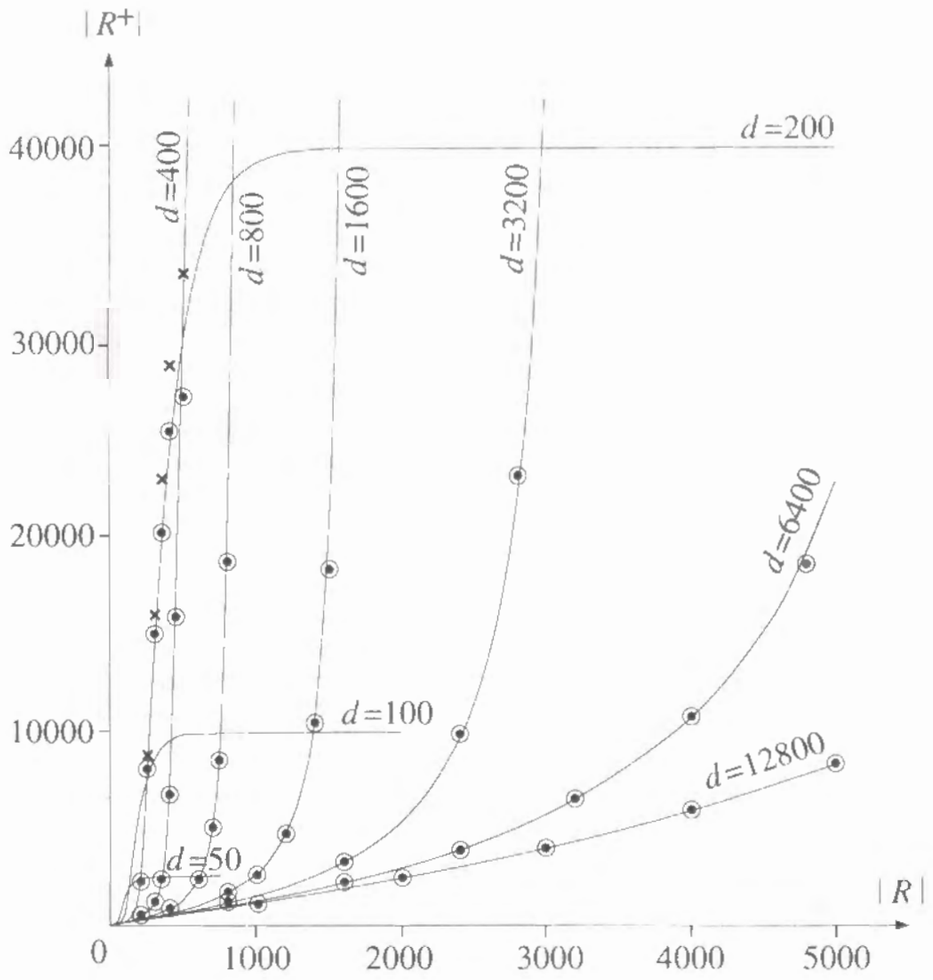


Figure 3.2: Estimation of $|R^+|$.

3.4 Application to the Same Generation Query

As an example for applying our estimation method, we choose the same generation relation query `sg(a,Y)` to output all persons who are the same generation as a given person `a`, which is well known in the deductive database applications.

By comparing the estimates with computational results, we shall see that our approach is useful for practical purposes. Therefore, our estimations can be used to assess the computational costs of various methods that answer a given query, and can compare them to pick up the most efficient one for the given problem instance.

3.4.1 Three Representative Methods

Definition 0 of the same generation predicate:

$$\begin{cases} \texttt{sg(X,X)} \leftarrow \\ \texttt{sg(X,Y)} \leftarrow \texttt{par(X,X1)}, \texttt{par(Y,Y1)}, \texttt{sg(X1,Y1)}. \end{cases} \tag{3.29}$$

Query:

$$\texttt{query(Y)} \leftarrow \texttt{sg(a,Y)}.$$

Here, par is an EDB predicate, while sg is an IDB predicate. We assume that the domains of two attributes in relation par are the same, and $d = 50$ and $|par| = 20$ in our example. We use this example throughout this section, and apply three representative methods to derive its answers.

Method 1: This method considers the following definitions which are equivalent to (3.29).

Definition 1 of the same generation predicate:

$$\begin{cases} \texttt{sg(X,X)} \leftarrow \\ \texttt{sg(X,Y)} \leftarrow \texttt{par} \times \texttt{par(X,Y,X1,Y1)}, \texttt{sg(X1,Y1)}. \end{cases}$$

This method generates a 4-ary relation $I_1\ (X, Y, X1, Y1) = par \times par(X, Y, X1, Y1)$ which is the Cartesian product of $par(X, X1)$ and $par(Y, Y1)$, and compute its transitive closure $I_1^+\ (X, Y, X2, Y2)$ by

$$\varphi(I_{1,i}) = \pi_{\{X,Y,X2,Y2\}}\Big(I_{1,i-1}(X,Y,X1,Y1) \bowtie I_1(X1,Y1,X2,Y2)\Big).$$

Then, it selects the tuples of I_1^+ whose third element coincides with the fourth element, that is, $I_2\ (X, Y, X2, Y2) = \sigma_{\{X2=Y2\}} I_1^+(X,Y,X1,Y1)$, and project them onto the first and second columns. The resulting relation is $sg(X,Y)$. Finally, we select the tuples whose first element is a from $sg(X,Y)$, which are the answer $sg(a,Y)$.

Estimation: First of all, the expected number of different values that appear in each attribute of a relation par is

$$d' = 50\left\{1 - \left(1 - \frac{1}{50}\right)^{20}\right\} = 16.62 \quad \text{(by (3.16))}.$$

The size of the transitive closure I_1^+ is

$$\Delta\rho_1 = |I_1| = 20 \times 20 = 400 \quad \text{(by (3.1))},$$
$$c = (16.62)^4 \quad \text{(by (3.18))},$$
$$p = \frac{400}{50 \times 50} = 0.16 \quad \text{(by (3.14))},$$
$$\Delta\rho_3 = 10.12 \quad \text{(by (3.25))}.$$

Therefore,

$$|I_1^+(X,Y,X2,Y2)| = 475.90 \quad \text{(by (3.26))}.$$

Then, the sizes of $I_2\ (X, Y, X2, Y2)$, $sg(X,Y)$ and $sg(a,X)$ are

$$|I_2(X,Y,X2,Y2)| = 475.90/16.62 = 28.63 \quad \text{(by (3.5))},$$

$$|sg(X,Y)| = 16.62^2\left\{1 - \left(1 - \frac{1}{16.62^2}\right)^{28.63}\right\} = 27.25 \quad \text{(by (3.6))},$$

$$|sg(a,Y)| = 27.25/16.62 = 1.64 \quad \text{(by (3.5))}.$$

As a result, the sum of the sizes of all intermediate relations (i.e., which can be considered as the processing cost) by Method 1 is

$$\text{Cost } 1 = |I_1(X,Y,X1,Y1)| + |I_1^+(X,Y,X2,Y2)|$$
$$+|I_2(X,Y,X2,Y2)| + |sg(X,Y)| + |sg(a,Y)| = 933.42.$$

■

Method 2: This method considers the following definitions, which are equivalent to (3.29).

Definition 2 of the same generation predicate:

$$\begin{cases} \texttt{sg(X1,Y1)} \leftarrow \texttt{par(X1,Z)}, \texttt{par(Y1,Z)} \\ \texttt{sg(X,Y)} \leftarrow \texttt{par(X,X1)}, \texttt{sg(X1,Y1)}, \texttt{par(Y,Y1)}. \end{cases}$$

This method first computes

$$sg1(X1,Y1) = \pi_{\{X1,Y1\}}\Big(par(X1,Z) \bowtie par(Y1,Z)\Big).$$

Then, it prepares the transitive closure $sg(X,Y) = sg1^+(X,Y)$ by repeating a single operation

$$\varphi(sg1_i) = \pi_{\{X,Y\}}\Big(sg1_{i-1}(X1,Y1) \bowtie (par(X,X1), par(Y,Y1))\Big).$$

Finally, $sg(a,Y) = \sigma_{\{X=a\}}\, sg(X,Y)$ gives the answer.

Estimation: First, we estimate the size of $sg1$ as follows:

$$|par(X,Y)| = 20, \quad d = 50,$$
$$d' = 16.62 \quad \text{(same as Method 1)},$$
$$c = 16.62^2 \quad \text{(by (3.18))},$$
$$|sg1(X,Y)| = 16.62^2\left\{1 - \left(1 - \frac{1}{16.62^2}\right)^{\frac{20\times 20}{16.62}}\right\}$$
$$= 23.09 \quad \text{(by (3.6) and (3.8))}.$$

Next, the size of the transitive closure $sg(X, Y)$ of $sg1(X, Y)$ and $sg(a, Y)$ becomes

$$\Delta\rho_1 = |sg1| = 23.09,$$
$$p = \frac{1}{50} \times \frac{1}{50} \times 20 \times 20 = 0.16 \quad \text{(by (3.14))},$$
$$\Delta\rho_3 = 0.49 \quad \text{(by (3.25))},$$
$$|sg(X,Y)| = 27.23 \quad \text{(by (3.26))},$$
$$|sg(a,Y)| = 27.23/16.62 = 1.64 \quad \text{(by (3.5))}.$$

Therefore, the processing cost is

$$\text{Cost } 2 = |sg1(X,Y)| + |sg(X,Y)| + |sg(a,Y)| = 51.95.$$

■

Method 3: The Magic Set method.

Definition 3 of the same generation predicate:

$$\begin{cases} \texttt{m-sg(X)} \leftarrow \texttt{m-sg(a)} \\ \texttt{m-sg(X)} \leftarrow \texttt{m-sg(X1)}, \texttt{par(X1,X)}, \end{cases}$$

$$\begin{cases} \texttt{sg(X1,Y1)} \leftarrow \texttt{m-sg(X1)}, \texttt{par(X1,Z)}, \texttt{par(Y1,Z)} \\ \texttt{sg(X,Y)} \leftarrow \texttt{m-sg(X)}, \texttt{par(X,X1)}, \texttt{sg(X1,Y1)}, \texttt{par(Y,Y1)}. \end{cases}$$

This method derives only the ancestors of a and a itself as $m\text{-}sg(X)$, which is defined by the first two formulas. Namely, we compute the transitive closure $sg(X,Y)$ by

$$\varphi_2(sg1') = \pi_{\{X,Y\}}\Big(sg1'(X1,Y1) \bowtie (par'(X,X1), par(Y,Y1))\Big), \tag{3.30}$$

where

$$sg1'(X1,Y1) = \pi_{\{X1,Y1\}}\Big(par'(X1,Z) \bowtie par(Y1,Z)\Big),$$

$$par'(X,X1) = m\text{-}sg(X) \bowtie par(X,X1).$$

Finally, we select from $sg(X,Y)$ the tuples with the first element $X = a$.

Estimation: We first remark that $m\text{-}sg(X)$ is the union of a and the transitive closure $m\text{-}sg1^+(X)$ defined by

$$\varphi_1(m\text{-}sg1) = \pi_{\{Y\}}\Big(m\text{-}sg1(X) \bowtie par(X,Y)\Big),$$

where $m\text{-}sg1(X) = \sigma_{\{X1=a\}}\, par(X1,\, X)$. As the size of $m\text{-}sg1(X)$ is

$$|\sigma_{\{X1=a\}} par(X1, X)| = 20/16.62 = 1.20 \quad \text{(by (3.5))},$$

we obtain

$$\Delta\rho_1 = 1.20,\ p = 20 \times \frac{1}{50} = 0.4,\ d' = c = 16.62,$$
$$\Delta\rho_3 = 0.16 \quad \text{(by (3.25))},$$
$$|m\text{-}sg1^+(X)| = 1.92 \quad \text{(by (3.26))},$$

and therefore

$$|m\text{-}sg(X)| = |m\text{-}sg1^+(X) \cup \{a\}|$$
$$= 1.92 + 1 - \frac{1.92 \times 1}{50} = 2.89 \quad \text{(by (3.3))}.$$

Then we compute par' by joining $m\text{-}sg(X)$ and $par(X, X1)$ by attribute X. But since $m\text{-}sg(X)$ is not generated randomly at this time, it may not be appropriate to apply formula (3.7). Therefore, as the tuples in $m\text{-}sg(X)$ of size 2.89, we consider element a (size 1) and other elements (size 1.89) separately, when the probability of being joined with $par(X, X1)$ is estimated, that is

$$|par'(X, X1)| = 1 \times 20 \times \frac{1}{16.62} + 1.89 \times 20 \times \frac{1}{50}$$
$$= 1.20 + 0.76 = 1.96. \tag{3.31}$$

In this formula, the first term in the righthand shows the expected value of the number of tuples in $par'(X,\, X1)$ with $X = a$, and the second term shows the expected value of the number of tuples in which X $(\neq a)$ is one of the ancestors of a.

Next, we evaluate the size of $sg1'$. Since the number of different values that appear as $X1$ in $par'(X1, Z)$ is the number of a and its ancestors, it becomes 2.89 which is computed as the size of $m\text{-}sg(X)$. Similarly, the number of different values of $Y1$ in $par(Y1, Z)$ is 16.62. By formulas (3.18), (3.8) and (3.6), therefore,

$$c = 2.89 \times 16.62 = 47.97,$$

$$|sg1'(X1, Y1)| = 47.97\left\{1 - \left(1 - \frac{1}{47.97}\right)^{\frac{1.96 \times 20}{16.62}}\right\} = 2.32.$$

By using these results, we can estimate the size of sg which is the transitive closure of par'. It is clear that

$$\Delta\rho_1 = |sg1'| = 2.32,$$
$$c = 47.97 \quad \text{(same as the previous } c\text{)}.$$

Then in order to estimate p for a single iteration in the transitive closure, we examine the probability for the $X1$ in $sg1'\ (X1, Y1)$ to be joined with the $X1$ in $par'(X1, Z)$, in formula (3.30). The element a and its proper ancestors appear at the ratio of 1.20 : 0.76 as the values of $X1$ in $sg1'$. This is the same as in par' (by (3.31)). Therefore, the probability of joining these elements with $X1$ in par' is

$$\frac{1.20}{1.96} \times \frac{1}{50} + \frac{0.76}{1.96} \times \frac{1}{1.92} = 0.21,$$

because the probability for a to be joined with $X1$ is equal to the probability that a appears as an ancestor of a, which is 1/50 from the uniform randomness of par, and the probability for a's ancestors to be joined with $X1$ is $1/|m\text{-}sg1^+(X)|$ by assuming that a's ancestors appear as $X1$ in par' with equal probability. On the other hand, the probability for $Y1$ in $sg1'$ to be joined with $Y1$ in par is 1/50 because of the randomness of par. From these facts, p can be estimated as

$$p = (0.21 \times 1.96) \times \left(\frac{1}{50} \times 20\right) = 0.17 \quad \text{(by (3.14))}.$$

This implies,

$$\Delta\rho_3 = 0.059 \quad \text{(by (3.25))},$$
$$|sg(X, Y)| = 2.77 \quad \text{(by (3.26))}.$$

Finally, since a and a's ancestors $(\neq a)$ should appear in X at the ratio of 1.20 : 0.76, in selecting the tuples of $X = a$ in $sg(X, Y)$, we have

$$|sg(a, Y)| = 2.77 \times \frac{1.20}{1.96} = 1.70.$$

Consequently, the total processing cost is

$$\text{Cost 3} = |m\text{-}sg(X)| + |par'(X1, Z)| + |sg1'(X1, Y1)|$$
$$+|sg(X, Y)| + |sg(a, Y)| = 8.08.$$

■

From the above discussion with the proposed three methods, we obtain the following estimations of processing costs,

Method 1: 932.52, Method 2: 51.95, Method 3: 8.08.

The differences among these costs are remarkable. Indeed, we recognize again that the magic set method is quite efficient. The final sizes of the answer set $sg(a, Y)$ is estimated as

Method 1: 1.64, Method 2: 1.64, Method 3: 1.70.

It should be noted that these three values are close to one another, although they are estimated as a result of applying several different operations.

From these facts, our approximations in this chapter may be said to be fairly accurate.

3.4.2 Numerical Experiments

To see the accuracy of our estimation in this section, we randomly generated [Knu 75] sample relations with $d = |D| = 50, 100, 200$ and several $r = |par(X, Y)|$ for each d, and computed the average sizes of intermediate relations that appear during executing Method 1 of this section. The results are shown in Tables 3.1, 3.2 and 3.3. Except that $|I_2|$ and $|sg(X, Y)|$ have some discrepancy between the real data and the estimated values, it appears that our approximations give rather accurate estimations in almost all cases.

r	Classification	$\lvert I_1\rvert$	$\lvert I_1^+\rvert$	$\lvert I_2\rvert$	$\lvert sg(X,Y)\rvert$	$\lvert sg(a,Y)\rvert$	Cost 1
20	Theory	400.00	475.90	28.63	27.25	1.64	933.42
	Experiment	400.00	471.40	39.40	24.05	1.50	936.35
30	Theory	900.00	1404.00	61.78	58.28	2.56	2426.62
	Experiment	900.00	1365.80	103.60	49.95	2.10	2421.45
40	Theory	1600.00	4410.81	159.15	143.83	5.19	6318.98
	Experiment	1600.00	5334.30	317.60	93.35	5.00	7350.25

Table 3.1: Results of Method 1 in case of $d = 50$.

r	Classification	$\lvert I_1\rvert$	$\lvert I_1^+\rvert$	$\lvert I_2\rvert$	$\lvert sg(X,Y)\rvert$	$\lvert sg(a,Y)\rvert$	Cost 1
40	Theory	1600.00	1904.47	57.53	56.07	1.69	3619.76
	Experiment	1600.00	1952.50	96.80	53.65	1.85	3704.80
50	Theory	2500.00	3332.54	84.37	82.15	2.08	6001.14
	Experiment	2500.00	3437.85	146.05	70.50	2.20	6156.60
60	Theory	3600.00	5622.71	124.16	120.51	2.66	9470.04
	Experiment	3600.00	5779.00	230.60	93.65	2.60	9705.85
70	Theory	4900.00	9600.21	190.04	183.17	3.63	14877.05
	Experiment	4900.00	10410.30	421.55	147.30	3.65	15882.80

Table 3.2: Results of Method 1 in case of $d = 100$.

r	Classification	$\|I_1\|$	$\|I_1^+\|$	$\|I_2\|$	$\|sg(X.Y)\|$	$\|sg(a.Y)\|$	Cost 1
40	Theory	1600.00	1666.63	45.87	45.10	1.24	3358.84
	Experiment	1600.00	1739.25	65.45	45.25	1.35	3451.30
60	Theory	3600.00	3955.94	76.15	75.10	1.45	7708.64
	Experiment	3600.00	4008.10	111.50	71.00	1.45	7792.05
80	Theory	6400.00	7618.75	115.31	113.82	1.72	14249.60
	Experiment	6400.00	7760.05	185.75	101.25	1.45	14448.50
100	Theory	10000.00	13332.53	169.10	166.83	2.12	23670.58
	Experiment	10000.00	13697.80	321.00	153.35	1.90	24174.65

Table 3.3: Results of Method 1 in case of $d = 200$.

3.5 Conclusion

In this chapter, we propose approximation formulas for estimating the sizes of resulting relations when typical operations in relational algebra and the operation of transitive closure are applied. Since these enable us to estimate the sizes of the resulting relations from the original sizes of original relations and domains, we can easily evaluate in advance the sizes of intermediate relations generated in processing deductive databases. This can be used to find efficient processing methods to derive answers to given database queries, without really executing the methods. In fact, we could indicate the differences among the three known methods for deriving answers of the same generation problem from given relations in Section 3.4.

Our initial assumptions on relations may not be appropriate in some cases, because

very few relations in the real world would have uniformly generated tuples in the precise sense. Therefore, we also have to consider the cases in which the probability distribution of values in the given domain is not uniform. In the next chapter therefore, we attempt to obtain estimation formulas of the expected sizes of resulting relations, when the original data are generated by general distributions.

Chapter 4

Approximate Evaluation of Processing Costs of General Data

4.1 Introduction

For the purpose of estimating the cost of processing queries in advance, we proposed a new approximation method in Chapter 3 to evaluate the size of transitive closure as well as the operations in relational algebra. These approximations were derived on condition that the value of each element of the original relations is randomly generated by the uniform probability distribution, independent of the values of other elements.

Although the above method gives a fairly accurate approximation within the range of uniform data, the distribution of real data seldom obeys the uniform distribution [Lyn 88, Wol 90]. Fortunately, in many cases, the probability distribution of the values in each domain of a relation is known in advance to a certain extent. To take a simple example, the distribution of family names in Japan is roughly known from the statistical data of names. Therefore, we suggested the necessity of handling the data that do not obey the uniform distribution in Chapter 3. The cost of processing queries can be evaluated more accurately if it is based on the real distribution of data values.

In this chapter, we assume that the distribution in each domain is known, or can be estimated by some means in advance. Then, we propose approximate methods for evaluating the computing cost of operations in relational algebra, especially transitive closures, whose definition was extended as presented in Chapter 3. The ideas to obtain approximate formulas are the refined version of those in the previous chapter, but several new ideas are incorporated in their construction. The result is more accurate even for the case of uniform distribution, as evidenced by some numerical experiments.

Finally, we take the Zipf distribution [Knu 73, Knu 75, Lyn 88, Wol 90] as a typical example, and demonstrate the effectiveness of our approximations by experiments. The Zipf distribution can represent fairly wide spectrum of distributions from uniform to very skewed ones.

4.2 Computing Cost of Operations in Relational Algebra

4.2.1 Assumptions on Relations and Functional Dependencies

We assume that the occurrence probability p_{ij} of value e_j in attribute A_i is known in advance or can be estimated by some means in this chapter. For example, by counting the number of occurrences of value e_j in attribute A_i of given relation R, we may regard that $p_{ij} \approx n_{ij}/r$, which may be sufficient for practical purposes. All entries of a given relation R, are generated independently according to such probability distribution. However, since we define that a relation does not contain the same tuple more than once, after generating tuples in the above manner, we remove all the duplicate tuples but one, and consider the remaining set of tuples forms a given relation R.

As a result of the above eliminations, the independence among the entries does not hold any longer, strictly speaking, and the probability distribution of the resulting values may be slightly distorted from the original distribution p_{ij}. But, for the ease of analysis, we consider and assume that the independency still holds and the distribution of values

still obey the original distribution p_{ij}.

Concerning with the independency assumption, there is a discussion saying that the independency does not hold because the relations we encounter in the real world have functional dependency [Ull 89b] among attributes. For example, in the parent-child relation $par(X, Y)$, attribute X (the child's family name) and attribute Y (the parent's family name) have the high probabilistic tendency that a child and his/her parent have the same family name (even if a female may change her family name after her marriage). This is an example in which X and Y are functionally dependent, and X and Y are not dependent. However, as another examples, let us consider the relation $flight(X, Y)$, which implies that there is a direct flight from the airport X to Y, and the relation $class(X, Y)$, which implies that a teacher Y teaches a subject X. In these relations, X and Y may be functionally dependent, since for example in the latter case, Y would determine X almost uniquely. However, it is difficult to believe that there are probabilistic dependency among airport names X and Y and subject X and teacher names Y. In other words, even if the values in each attribute are randomly generated independently of the values in the other attributes, a functional dependency between those attributes can hold. These tell that the probabilistic independence and the functional dependency are not contradictory each other. In this sense, even if the assumption of independency is made, the results will supply useful information for most of relations practically encountered. On the other hand, it is also true that the cost analysis of operations is very difficult without the assumption of independency.

There is another factor to be considered in our analysis. That is, even if we assume that a relation R is generated in the above manner, the probability distribution will be modified whenever any operation is applied to R. This is quite different from the case of uniform distribution, because the uniform distribution is maintained after applying many of the operations under consideration. Therefore, in the analysis of this section,

we carefully consider how the distribution changes while estimating the cost of various operations.

4.2.2 Set Operations

Corresponding to the results in Section 2 of Chapter 3, we present the approximate formulas for the computing cost of basic operations in relational algebra under the general probabilistic distribution of values in each attribute.

First, the size of the Cartesian product $R_1 \times R_2$ is independent of the distribution, and is

$$|R_1 \times R_2| = r_1 r_2. \tag{4.1}$$

Let k-ary relations R_1 and R_2 have attributes A_i and B_i $(i = 1, \ldots, k)$, respectively, where A_i and B_i have the same domain D_i. Let the occurrence probability of the value e_j in A_i and B_i be p_{ij} and q_{ij}, respectively. Then, the expected size of their intersection is

$$|R_1 \cap R_2| \approx \sum_{j_1=1}^{d_1} \cdots \sum_{j_k=1}^{d_k} \left\{1 - (1 - p_{1j_1} \cdots p_{kj_k})^{r_1}\right\}\left\{1 - (1 - q_{1j_1} \cdots q_{kj_k})^{r_2}\right\}. \tag{4.2}$$

When $p_{1j_1} \cdots p_{kj_k}$ and $q_{1j_1} \cdots q_{kj_k}$ are sufficiently smaller than 1, we can linearize the above formula as

$$|R_1 \cap R_2| \approx r_1 r_2 \cdot \sum_{j_1=1}^{d_1} \cdots \sum_{j_k=1}^{d_k} p_{1j_1} \cdots p_{kj_k} q_{1j_1} \cdots q_{kj_k}$$

in practice. Especially, when the values in domain D_i are generated by the uniform distribution, that is, $p_{ij_1} = q_{ij_1} = 1/d_i$ $(i = 1, \ldots k,\ j_1 = 1, \ldots, d_i)$, the above formula becomes

$$\begin{aligned} |R_1 \cap R_2| &\approx r_1 r_2 \cdot \sum_{j_1=1}^{d_1} \cdots \sum_{j_k=1}^{d_k} \left(\frac{1}{d_1 \cdots d_k}\right)^2 \\ &= r_1 r_2 \cdot \frac{1}{d_1 \cdots d_k}, \end{aligned}$$

as already presented in Chapter 3.

The sizes of union and set difference can be obtained from the size of intersection. that is,

$$|R_1 \cup R_2| = r_1 + r_2 - |R_1 \cap R_2| \tag{4.3}$$

and

$$|R_1 - R_2| = r_1 - |R_1 \cap R_2|. \tag{4.4}$$

4.2.3 Selection, Projection and Natural Join

The expected size of projection $\pi_C R$ with the condition that

$$C = \{A_{i_1} = e_{j_1}, \ldots, A_{i_m} = e_{j_m}\}$$

is approximately equal to

$$|\pi_A R| \approx \sum_{j_1=1}^{d_{i_1}} \cdots \sum_{j_k=1}^{d_{i_k}} \left\{1 - (1 - p_{i_1 j_1} \cdots p_{i_k j_k})^r\right\}. \tag{4.5}$$

Similarly, the expected size of selection σ_A onto the set of attributes

$$A = \{A_{i_1}, \ldots, A_{i_m}\}$$

is

$$|\sigma_C R| \approx p_{i_1 j_1} \cdots p_{i_k j_k} \times r. \tag{4.6}$$

Now, let R_1 and R_2 have attributes $A_1, \ldots, A_m$ and $B_1, \ldots, B_n$, respectively, and the occurrence probability of value e_j in the attribute A_i (B_i) of R_1 (R_2) be p_{ij} (q_{ij}). Then, the expected size of the natural join with respect to some of attributes $A_{i_1}, \ldots, A_{i_c}$ in R_1 and $B_{j_1}, \ldots, B_{j_c}$ in R_2, where the domains of A_{i_s} and B_{j_s} coincide, becomes

$$|R_1 \bowtie R_2| \approx r_1 r_2 \prod_{s=1}^{c} \sum_{m=1}^{d_{i_s}} p_{i_s m} \cdot q_{j_s m}. \tag{4.7}$$

In other words, we estimated the selectivity s_{12} between R_1 and R_2 as

$$s_{12} = \prod_{s=1}^{c} \sum_{m=1}^{d_{i_s}} p_{i_s m} \cdot q_{j_s m}.$$

As a special case, which is important in practice, let us consider the natural join of $R(A_1, A_2)$, where the domains of the two attributes A_1 and A_2 are the same D. Furthermore, let the occurrence probability of value e_j in A_1 and A_2 be p_{1j} and p_{2j}, respectively. Then, the expected size of the resulting relation is

$$\left| R \underset{A_2=A_1}{\bowtie} R \right| \approx r \times r \times \sum_{j=1}^{d} p_{1j} q_{2j}.$$

4.3 Computing Cost of Transitive Closure

4.3.1 Outline of the Estimation

In this section, we derive an approximate formula for estimating the size of a transitive closure by a similar approach as used in Chapter 3.

Recall that the Semi-Naive algorithm for the generalized transitive closure is summarized in the following computation:

$$R_i = \pi_{A^{(0)}} \left(\Delta R_i \underset{R.A^{(1)}=Q_1.A^{(1)},\ldots,R.A^{(u)}=Q_u.A^{(u)}}{\bowtie} (Q_1, \ldots, Q_u) \right).$$

In this computation, we first initialize the intermediate relations ΔR_1 and R_1 as

$$\begin{cases} \Delta R_1 := R, \\ R_1 := \Delta R_1. \end{cases} \tag{4.8}$$

Then, the following computation is iterated for $i = 2, 3, \ldots$,

$$\begin{cases} R'_i := \Delta R_{i-1} \bowtie (Q_1, \ldots, Q_u) \\ \delta R_i := \pi_{A^{(0)}} R'_i \\ \Delta R_i := \delta R_i - R_{i-1} \\ R_i := R_{i-1} \cup \Delta R_i. \end{cases} \tag{4.9}$$

Finally, the transitive closure R^+ of R by $Q_1, \ldots, Q_u$, $C = \{R.A^{(1)} = Q_1.A^{(1)}, \ldots, R.A^{(u)} = Q_u.A^{(u)}$ and $A^{(0)}$ is computed by

$$R^+ \left[Q_1, \ldots, Q_u; C; A^{(0)} \right] = \bigcup_{k=1}^{\infty} R_i \quad (R_1 = R).$$

We estimated in Chapter 3 the size of the generalized transitive closure by decomposing the computation in each iteration into the operations of relational algebra. In this case, operations such as natural join, projection, set difference and union are used. In other words, we can estimate the expected size of the transitive closure if we can know the sizes of relations and the probability distribution of values after such operations.

If all the values of all attributes are generated by the uniform distribution, the uniform distribution is preserved after applying the above operations to the original relations, and we could analyze their behaviors rather easily as described in Chapter 3. In the case of general probability distribution, however, the distribution changes after applying each operation to relations. Since it is not easy to trace how the probability distribution changes, we use approximate distribution in our computation. As it is often the case that the probability distributions in the original relations $R\ (= \Delta R_1)$ and $Q_1, \ldots, Q_u$ are already given by approximation, this simplified treatment may be sufficient for practical purposes.

The initial relations $\Delta R_1 = R_1 = R$ as well as the resulting relations R_i after immediate or a few iterations still maintain the characteristics of the original distribution. As the iteration of computing R_i progresses, however, probability distribution of values will become more spread out and will approach to the uniform distribution eventually. This is because we eliminate duplicate tuples at each iteration, and add the set of new tuples ΔR_i to R_i. As a result of these modifications, each value tends to appear uniformly.

For the purpose of estimating the accurate sizes of relations R_i for general data, we have to trace the changes of probability distribution of values for every operations, as we mentioned. If it can be possible, we should trace them completely after each iteration. But it is too complicated to do so in practice. Therefore, in the subsequent discussion in this section, we estimate the size of relations R_i approximately by dividing the entire computation into the following two phases:

- The first iteration which is based on the original probability distribution of R, and
- The second and higher iterations which are based on the uniform distribution of R_1.

4.3.2 First Iteration

First of all, consider relations R and $Q_1, \ldots, Q_u$, and assume that the occurrence probability of value e_j in attribute A_i $(i = 1, \ldots, k)$ of R is p_{ij} $(j = 1, \ldots, d_i)$ and the occurrence probability of value e_j in attribute $A_i^{(h)}$ of Q_h $(h = 1, \ldots, u)$ is $t_{ij}^{(h)}$.

The computation in the first iteration is obtained by substituting $i = 2$ in (4.9), such as

$$\begin{cases} R_2' = \Delta R_1 \bowtie (Q_1, \ldots, Q_u) \\ \delta R_2 = \pi_B R_2' \\ \Delta R_2 = \delta R_2 - R \\ R_2 = R_1 \cup \Delta R_2. \end{cases} \tag{4.10}$$

We estimate the expected sizes of R_2', δR_2, ΔR_2 and R_2 in the above formulas in this order.

The size of R_2'

The probability that all the join attributes $A_i^{(h)} \in A^{(h)}$ in each relation Q_h (whose size is q_h) are joined with the corresponding attributes in $\Delta R_1 (= R)$, that is, the selectivity f_h between R and Q_h, becomes

$$s_h = \prod_{A_i^{(h)} \in A^{(h)}} \sum_{j=1}^{d_i} p_{ij} \cdot t_{ij}^{(h)} \quad (h = 1, \ldots, u).$$

Therefore, by using the formula (3.7) of join iteratively, we obtain

$$|R_2'| = r \times s_1 q_1 \times \cdots \times s_u q_u.$$

In other words, the selectivity s between R and $Q_1, \ldots, Q_u$ is

$$s = \prod_{i=1}^{u} f_i.$$

The size of δR_2

The relation δR_2 is obtained by projecting R_2' onto the set of attributes $A^{(0)}$. In this process, we use the new domain sizes of $A_i \in A^{(0)}$,

$$d_i' = \sum_{j=1}^{d} \left\{ 1 - (1 - p_{ij})^r \right\} \quad (i = 1, \ldots, v),$$

$$d_i' = \sum_{j=1}^{d} \left\{ 1 - (1 - t_{ij}^{(h)})^{q_h} \right\} \quad (i = v+1, \ldots, m),$$

because only those values which have already appeared in the attributes A_i in $A^{(0)}$ of the original R and $Q_1, \ldots, Q_u$ can appear in the new attributes A_i in R_2'. The first formula expresses the expected number of different values in attribute A_i $(\in A^{(0)})$ in R and the second one expresses that of attribute A_i $(\in A^{(0)})$ in Q_i's.

It is not so easy to know that the exact probability distribution of these d_i' different values because we have to consider all the combinations of possible values appeared in the resulting relation. Therefore, for simplicity, we suppose that these d_i' different values are generated by the following probability distributions p_{ij} and $t_{ij}^{(h)}$, which are slightly modified from the original probability distribution p_{ij} in R. and $t_{ij}^{(h)}$ in Q_h $h = 1, \ldots, u$:

$$p_{ij}' = \frac{p_{ij}}{\sum_{j=1}^{\lceil d_i' \rceil} p_{ij}}, \quad t'^{(h)}_{ij} = \frac{t_{ij}^{(h)}}{\sum_{j-1}^{\lceil d_i' \rceil} t_{ij}^{(h)}} \quad (j = 1, \ldots, \lceil d_i' \rceil).$$

These probability distributions assume that only the first $\lceil d_i' \rceil$ values with high occurrence probability (we consider that $e_{i_1}, \ldots, e_{i_{\lceil d_i' \rceil}}$ are the first $\lceil d_i' \rceil$ values without loss of generality) are generated in the new relation by the normalized distributions of the original distributions.

In conclusion, we can estimate the expected size of δR_2 by using those modified distributions as

$$|\delta R_2| = \sum_{j_1=1}^{\lceil d'_1 \rceil} \cdots \sum_{j_v=1}^{\lceil d'_v \rceil} \sum_{j_{v+1}=1}^{\lceil d'_{v+1} \rceil} \cdots \sum_{j_m=1}^{\lceil d'_m \rceil} \left\{ 1 - \left(1 - p'_{1j_1} \cdots p'_{vj_v} \cdot t'^{(h)}_{v+1j_{v+1}} \cdots t'^{(h)}_{mj_m} \right)^{|R'_2|} \right\}$$
$$(h \in \{1, \ldots, u\}).$$

The size of ΔR_2

The relation ΔR_2 is obtained by the set difference, that is, it consists of only newly generated tuples in the first iteration. Therefore, the size of ΔR_2 becomes

$$\begin{aligned} |\Delta R_2| &= |\delta R_2| - \sum_{j_1=1}^{\lceil d'_1 \rceil} \cdots \sum_{j_m=1}^{\lceil d'_m \rceil} \left\{ 1 - \left(1 - p'_{1j_1} \cdots p'_{vj_v} p'_{v+1j_{v+1}} \cdots p'_{mj_m} \right)^r \right\} \\ &\qquad \cdot \left\{ 1 - \left(1 - p'_{1j_1} \cdots p'_{vj_v} t'^{(h)}_{v+1j_{v+1}} \cdots t'^{(h)}_{mj_m} \right)^{|\delta R_2|} \right\} \\ &\approx |\delta R_2| - r \cdot |\delta R_2| \cdot \sum_{j_1=1}^{\lceil d'_1 \rceil} \cdots \sum_{j_m=1}^{\lceil d'_m \rceil} \left(p'^2_{1j_1} \cdots p'^2_{vj_v} \right. \\ &\qquad \left. \cdot\, p'_{v+1j_{v+1}} t'_{v+1j_{v+1}} \cdots p'_{mj_m} t'_{mj_m} \right). \end{aligned} \tag{4.11}$$

The size of R_2

Finally, we have to estimate the size of R_2, which is the union of R_1 $(= \Delta R_1 = R)$ and ΔR_2. Here, we know that there are no tuples that belong to both R_1 and ΔR_2 because of the definition of ΔR_1. Therefore, we obtain

$$\begin{aligned} |R_2| &= |R_1| + |\Delta R_2| \\ &= r + |\Delta R_2|. \end{aligned}$$

This concludes the computation of the first iteration. For the same reason as we mentioned in Chapter 3, we use notations ρ'_i, $\delta\rho_i$, $\Delta\rho_i$ and ρ_i in place of the approximate sizes of R'_i, δR_i, ΔR_i and R_i, respectively, in order to emphasize that these are approximations.

4.3.3 Second and Higher Iterations

In the second and higher iterations ($i \geq 3$), the probability distribution of values in each attribute A_i of the incremental relation ΔR_i asymptotically approaches from the original distribution of relation R to the uniform distribution as the number of iterations i increases. To take into this effect, we consider that the distribution of values in all the relations R'_i, δR_i, ΔR_i and R_i in the second and higher iterations to be uniform, whose domains are further restricted from the d'_i's in the previous section to

$$d''_i = \sum_{j=1}^{\lceil d'_i \rceil} \left\{ 1 - (1 - p'_{ij})^{\Delta\rho_2} \right\} \quad (i = 1, \ldots, v),$$

$$d''_i = \sum_{j=1}^{\lceil d'_i \rceil} \left\{ 1 - (1 - t'^{(h)}_{ij})^{\Delta\rho_2} \right\} \quad (i = v+1, \ldots, m).$$

Then, in the computation of $R'_i = \Delta R_{i-1} \bowtie (Q_1, \ldots, Q_u)$, the selectivity s'_h between R_{i-1} and each Q_h (which is joined with the set of attributes $A^{(h)}_i$) becomes

$$s'_h = \prod_{A^{(h)}_i \in A^{(h)}} \sum_{j-1}^{\lceil d''_i \rceil} \frac{1}{\lceil d''_i \rceil} \cdot t^{(h)}_{ij} \quad (h = 1, \ldots, u),$$

because we assumed that all the values in attribute A_i of ΔR_{i-1} obey the uniform distribution of $\lceil d''_i \rceil$ different values. Then, the total selectivity s' between ΔR_{i-1} and $Q_1, \ldots, Q_u$ is

$$s' = \prod_{h=1}^{u} s'_h.$$

Finally, we can get the sizes for iterations $i \geq 3$ as

$$\begin{cases} \rho'_i = \Delta\rho_{i-1} \times s'_1 q_1 \times \cdots \times s'_u q_u = \Delta\rho_{i-1} \times s' \times \sum_{j-1}^{u} q_j \\ \delta\rho_i = d''_1 \cdots d''_m \left\{ 1 - \left(1 - \dfrac{1}{d''_1 \cdots d''_m} \right)^{\rho'_i} \right\} \\ \Delta\rho_i = \delta\rho_i - \dfrac{\delta\rho_i \times \rho_{i-1}}{d''_1 \cdots d''_m} \\ \rho_i = \rho_{i-1} + \Delta\rho_i. \end{cases} \tag{4.12}$$

Throughout the remaining iterations, we use these approximations. In these formulas, by letting

$$d^* = d_1'' \cdots d_m'',$$
$$F = s_1' q_1 \times \cdots \times s_u' q_u,$$

and by considering that $d_1'' \cdots d_m'' \gg 1$ usually holds, we can substitute the first formula of (4.12) into the second formula and derive the approximate formula

$$\begin{aligned}\delta\rho_i &\approx d^* \left\{1 - \left(1 - \frac{1}{d^*}\right)^{\Delta\rho_{i-1}\cdot F}\right\} \\ &\approx \Delta\rho_{i-1}\cdot F - \frac{1}{2d^*}\Big(\Delta\rho_{i-1}\cdot F\Big)\Big(\Delta\rho_{i-1}\cdot F - 1\Big) \\ &\approx \frac{F^2}{2d^*}\left(\frac{2d^*}{F} - \Delta\rho_{i-1}\right)\cdot\Delta\rho_{i-1}\end{aligned}$$

where we use only the first and second terms of the Taylor expansion of $\left(1 - \frac{1}{d^*}\right)^{\Delta\rho_{i-1}\cdot F}$. Then, by substituting this $\delta\rho_i$ into the third formula in (4.12), we obtain the following recursive equations

$$\begin{cases} \Delta\rho_i \approx \dfrac{F^2}{2d^{*2}}\left(\dfrac{2d^*}{F} - \Delta\rho_{i-1}\right) \times \Big(d^* - \rho_{i-1}\Big)\ \Delta\rho_{i\ -1} \\ \rho_i = \rho_{i-1} + \Delta\rho_i. \end{cases} \tag{4.13}$$

Consequently, solving this recursive equation leads us to

$$\begin{aligned}\rho^+ &\approx \sum_{i=1}^{\infty}\Delta\rho_i = \Delta\rho_1 + \tilde{\rho} \\ \tilde{\rho} &= \sum_{i=2}^{\infty}\Delta\rho_i,\end{aligned} \tag{4.14}$$

which gives an approximate size of the transitive closure R^+.

4.3.4 Solving the Recursive Equation Approximately

Similar to the case of the uniform distribution in Chapter 3, it is not easy to solve recursive equation (4.13) exactly. Therefore, we try to solve it approximately.

Let us transform the first formula in (4.13) into

$$\frac{2d^{*2}}{F^2}\cdot\Delta\rho_i \approx \frac{2d^{*2}}{F}\cdot\Delta\rho_{i-1} - \frac{2d^*}{F}\cdot\rho_{i-1}\cdot\Delta\rho_{i-1}$$

$$- d^*\cdot\Big(\Delta\rho_{i-1}\Big)^2 + \rho_{i-1}\cdot\Delta\rho_{i-1}\cdot\Delta\rho_{i-1},$$

where

$$C_1 = \frac{2d^{*2}}{F^2},\quad C_2 = \frac{2d^{*2}}{F},\quad C_3 = \frac{2d^*}{F},\quad C_4 = d^*.$$

Then, by summing them up for $i = 2, 3, \ldots$, and using C_1, C_2, C_3 and C_4, we obtain

$$\begin{aligned}C_1\,(\tilde{\rho} - \Delta\rho_2) \approx C_2\cdot\tilde{\rho} - \frac{1}{2}C_3\left(\tilde{\rho}^2 + \sum_{i=2}^{\infty}(\Delta\rho_i)^2\right) \\ - C_4\sum_{i=2}^{\infty}(\Delta\rho_i)^2 + \sum_{i=2}^{\infty}(\rho_i\cdot\Delta\rho_i\cdot\Delta\rho_i).\end{aligned}$$

In this formula, we used the following three approximations:

$$\sum_{i=2}^{\infty}(\rho_i\cdot\Delta\rho_i\cdot\Delta\rho_i) \approx \frac{1}{2}\left(\tilde{\rho}\sum_{i=2}^{\infty}(\Delta\rho_i)^2 + \sum_{i=2}^{\infty}(\Delta\rho_i)^3\right),$$

$$\sum_{i=2}^{\infty}(\Delta\rho_i)^2 \approx \tilde{\rho}\cdot\Delta\rho_3,$$

$$\sum_{i=2}^{\infty}(\Delta\rho_i)^3 \approx \tilde{\rho}\cdot(\Delta\rho_3)^2$$

to obtain

$$\begin{aligned}C_1\Big(\tilde{\rho} - \Delta\rho_2\Big) = C_2\cdot\tilde{\rho} - \frac{1}{2}\cdot C_3\Big(\tilde{\rho}^2 + \tilde{\rho}\cdot\Delta\rho_3\Big) \\ - C_4\cdot\tilde{\rho}\cdot\Delta\rho_3 + \frac{1}{2}\left(\tilde{\rho}^2\cdot\Delta\rho_3 + \tilde{\rho}\cdot\Big(\Delta\rho_3\Big)^2\right),\end{aligned}$$

where

$$\Delta\rho_3 = \frac{F^2}{2d^{*2}}\cdot\left(\frac{2d^*}{F} - \Delta\rho_2\right)\cdot(d^* - \Delta\rho_2)\cdot\Delta\rho_2,$$

and $\Delta\rho_2$ is computed by (4.11). Therefore, by solving the above quadratic equation in $\tilde{\rho}$, we have the solution

$$\begin{cases} \tilde{\rho} = \dfrac{-C + \sqrt{C^2 + 8C_1\Big(C_3 - \Delta\rho_3\Big)\Delta\rho_2}}{2\Big(C_4 - \Delta\rho_3\Big)}, \\ C = 2C_1 - 2C_2 + C_3\cdot\Delta\rho_3 + 2C_4\cdot\Delta\rho_3 - \Big(\Delta\rho_3\Big)^2. \end{cases} \tag{4.15}$$

Finally, the expected size ρ^+ of the transitive closure $R^-\left[Q_1,\ldots,Q_u;C;A^{(0)}\right]$ is approximately estimated by

$$\rho^+ \approx \Delta\rho_1 + \bar{\rho}. \tag{4.16}$$

4.4 Examples of Approximate Evaluations of the Sizes of Transitive Closure

In this section, we give some numerical examples of our results in this chapter, that is, we compute the sizes of transitive closures for some probability distributions.

4.4.1 Zipf Distribution

We consider the so-called Zipf distribution [Knu 73, Knu 75, Lyn 88, Wol 90] as a concrete example of a general distribution of values.

The Zipf distribution is defined as follows:

Definition 4.1: (Pure Zipf distribution) For a given domain of size n, the probability of value e_j is

$$p_j = \frac{1}{jH_n}, \quad j = 1,\ldots,n, \tag{4.17}$$

where

$$H_n = \sum_{j=1}^{n} \frac{1}{j}.$$

■

It is known that this distribution is commonly observed in the frequency of words in the sentences or the frequency of family names in one nation, and so on. Zipf distribution is often extended into the following form:

Definition 4.2: (Generalized Zipf distribution) For a given domain of size n, the probability of value e_j is

$$p_j = \frac{1}{j^{1-\theta}H_n^{(1-\theta)}}, \quad j = 1,\ldots,n, \tag{4.18}$$

where

$$H_n^{(1-\theta)} = \sum_{j=1}^{n} \frac{1}{j^{1-\theta}},$$

and θ is a parameter that represents the skewness of the distribution: $\theta = 0$ means the pure Zipf distribution and $\theta = 1$ means the uniform distribution. ■

4.4.2 Numerical Experiments on Zipf Distribution

Let us compute the transitive closure $R^+ = \bigcup_{i=1}^{\infty} R_i$ such that

$$\begin{cases} R_i(X,Y) = \pi_{\{X,Y\}}\Big(R_{i-1}(X,Y) \bowtie R(Y,Z)\Big), \\ R_1(X,Y) = R(X,Y). \end{cases}$$

We suppose that attributes X and Y have the same domain D of size d, and consider that the values in X and Y are generated by the Zipf distribution of

1. $\theta = 0$, i.e., the pure Zipf distribution, and
2. $\theta = 1$, i.e., the uniform distribution

in Definition 4.2. For the domain of size d and the relation of size r, we compute the following two kind of values:

1. The average size of transitive closures $|R^+|$, which is empirically observed by generating 30 relations randomly and by computing the true sizes of the transitive closures, and
2. ρ^+ of approximate formulas (4.15) and (4.16).

Figure 4.1 shows the above results in case of $d = 100$ and Figure 4.2 shows the results in case of $d = 400$. For the Zipf distribution of $\theta = 1$ (i.e., uniform distribution), we also show the approximate value ρ^+ obtained by the formula (3.28) in Chapter 3. for the

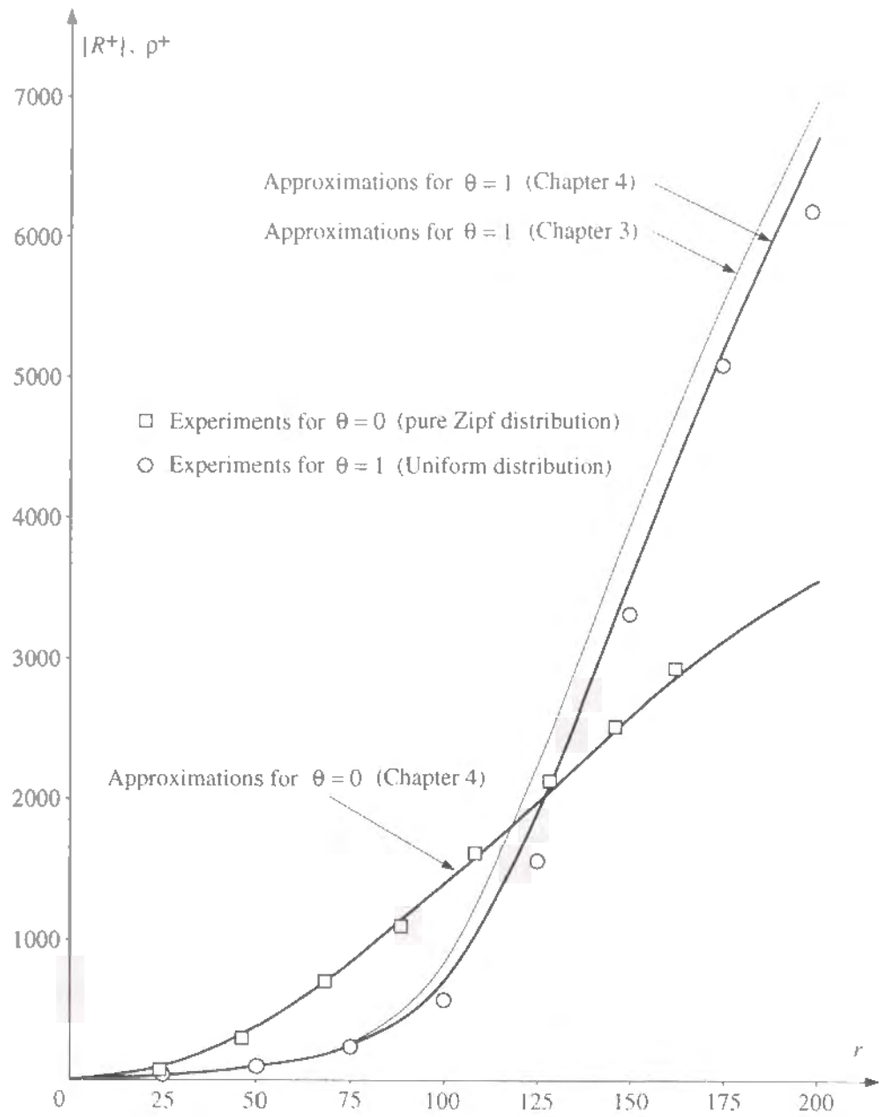


Figure 4.1: Sizes of transitive closures as a function of r ($d = 100$).

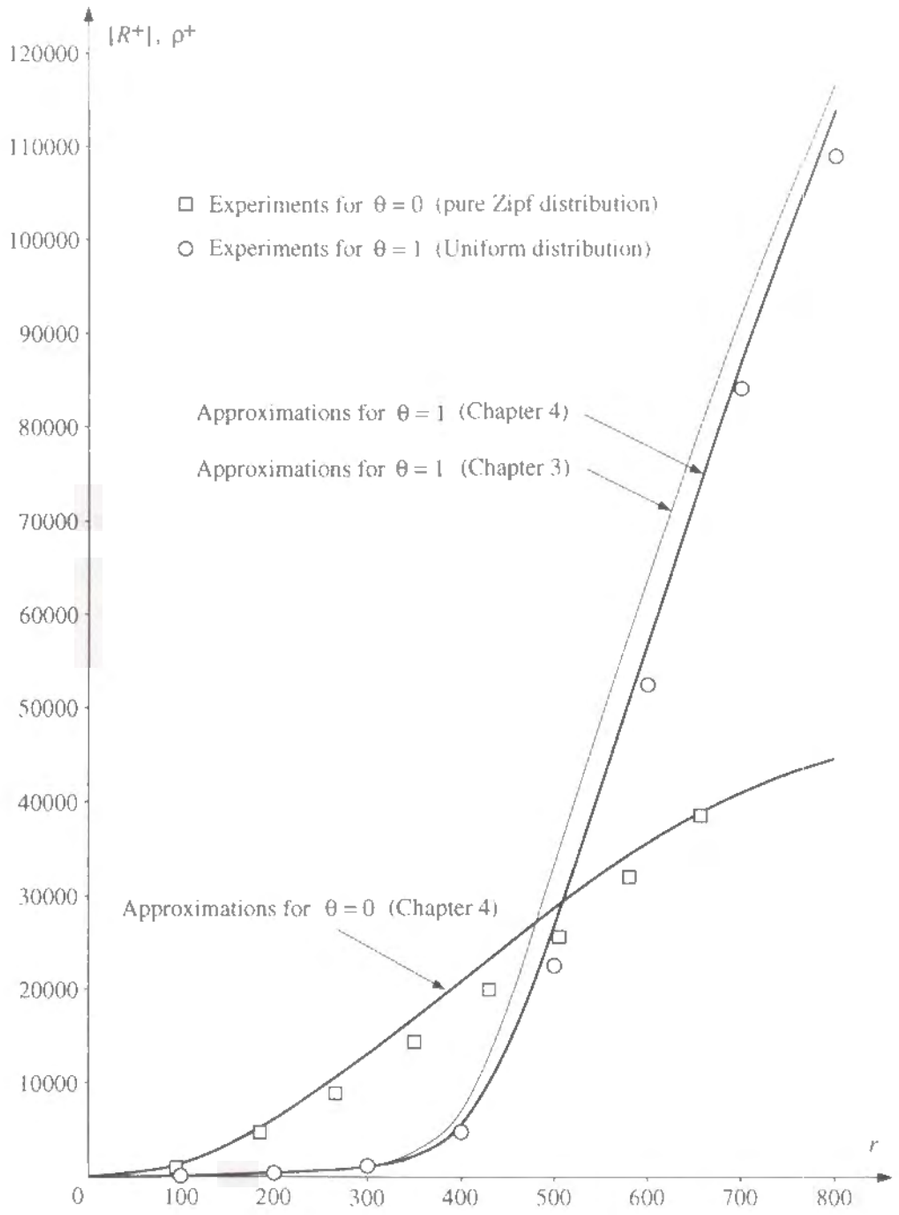


Figure 4.2: Sizes of transitive closures as a function of r ($d = 400$).

purpose of comparing the accuracies of two approximations proposed in Chapter 3 and this chapter.

In these experiments, we first observe that the sizes of the transitive closures in case of $\theta = 0$ considerably differ from those in case of $\theta = 1$, in both cases of $d = 100$ and $d = 400$. This implies that the size of a transitive closure is greatly influenced by the probability distribution of values in attributes of the relations. Therefore, the analysis that takes into account the probability distribution is indispensable and important.

Our approximation appears fairly accurate even in the case of $\theta = 0$, as we can see that ρ^+ is very close to the experimental values, although we employed rather rough approximations to obtain the formula. Furthermore, our approximation for $\theta = 1$ turned out to be more accurate than the approximation in Chapter 3. This is because we adopted the quadratic approximation of Taylor expansion in place of the linear approximation in solving the recursive equation (4.13).

As exhibited in the above numerical results, our proposal appears to be sufficiently accurate in the two extreme cases of probability distributions with $\theta = 0$ and 1. This indicates that the proposed formula may give us rather good approximations for a wide class of general distributions.

4.5 Conclusion

In this chapter, we firstly propose a new approximate method of estimating the computing cost of operations in relational algebra, when the values in attributes of the original relations are independently generated according to general probability distributions. Then, by using those results, we obtained an approximation formula for the sizes of transitive closures, by considering carefully the changes of probability distribution in each iteration from the original distribution. However, it appears impossible to know the exact probability distribution of values after each iteration, and we have to notice that there

is a theoretical limit if we try to extend our approach in their section in order to obtain better approximation formulas. To attain more accuracy, therefore, we need to construct another completely different approach.

Chapter 5

The Expected Size of Transitive Closure of a Random Digraph

5.1 Introduction

We present an approach to compute the expected size of transitive closures of relations in this chapter. This method is based on the theory of random graphs [Bol 85].

As we mentioned in Chapter 2, the transitive closures of relations are expressed and defined by directed graphs (or digraphs). Random directed graphs are usually specified by two parameters, the number of vertices n and the probability p that there is an arc from one vertex to another vertex. Although several kinds of studies have been made on random graphs, most of them are focused on random undirected graphs [Bol 82, Bol 85], and obtain very few results on random digraphs [Pit 83, Wri 73, Wri 78]. The theories and discussions for random digraphs must be very different from those for random undirected graphs.

For the purpose of estimating the expected size of transitive closures of random digraphs, we show firstly that it is sufficient to know the probability that has at least one path from an arbitrary vertex s to a vertex t ($\neq t$). We call this probability the reacha-

bility from s to t. We propose two methods for computing the exact reachability, after describing some properties about the reachability.

Then, we modify one of the methods into those for computing the approximate reachabilities easily, and provide some experimental results to examine their performance. In addition to this, by utilizing the exact reachability of the other method, we obtain simple lower and upper bounds on the reachabilities. Finally, we analyze the accuracy and the asymptotic performance of the upper bound on the reachability, the number of reachable vertices from a given vertex and the size of transitive closure of a random digraph.

5.2 Random Graph and its Properties

We present some basic notations, definitions and properties of random graphs in this section.

5.2.1 Random Graph

A random graph G is classified from the following two viewpoints; whether it is

1. undirected, or
2. directed,

and whether it contains

a. self-loops, or

b. no self-loops.

We denote a random graph G of these four types 1a, 1b, 2a and 2b by U_L, U, D_L and D, respectively, i.e., as shown in the table below.

	Undirected	Directed
Without loops	$U(n,p)$	$D(n,p)$
With loops	$U_L(n,p)$	$D_L(n,p)$

In all these cases, we assume that the graphs has no multiple edges or arcs, according to the conventions.

In discussioning the properties of random graphs, in general, there are two representative models of random graphs. We use the following model throughout this chapter.

Definition 5.1: A random graph G, which is made from either one of the types of U_L, U, D_L or D, is defined by the number of vertices n and the probability p of the presence of an arc from any vertex s to any vertex t. This probability is given independently of the presence or the absence of an arc between all the other pairs of vertices s' and t'. ■

If the type of a random graph is obvious from the context, we often eliminate it and simply denote the graph by $G(n,p)$, and we also use it for expressing all types of random graphs.

We show in Figure 5.1 these four random graphs G of types U, U_L, D and $D_L(n,p)$, with $n = 4$ and $p = 1$ (i.e., all possible edges (arcs) are present).

Now, let N be the number of all the possible pairs of vertices where an edge or an arc can exist in a random graph G for each type, that is,

$$N = \begin{cases} n(n-1)/2 & (\text{for } U) \\ n(n+1)/2 & (\text{for } U_L) \\ n(n-1) & (\text{for } D) \\ n^2 & (\text{for } D_L). \end{cases}$$

On account of the independence of the probability p, the expected number of edges or arcs m in a random graph G is considered to be $m = pN$.

On the other hand, there is another usual definition of random graphs. In this model, a random graph G is determined by the number of vertices n and the number of edges or arcs m, denoted by $G(n,m)$. Therefore, it has $\binom{N}{m}$ events to occur, and all the $\binom{N}{m}$ possible graphs are supposed to be generated with the same probability $1 \Big/ \binom{N}{m}$. As a

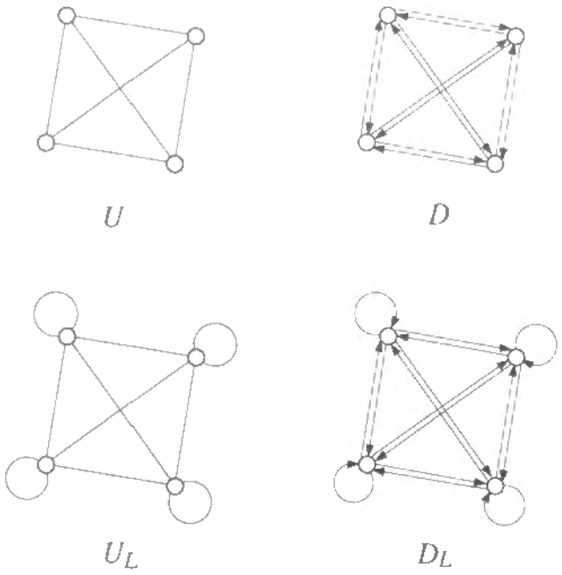


Figure 5.1: Four types of graphs with all possible edges or arcs.

result, the probability of having an edge or an arc between any two vertices eventually becomes

$$p = \frac{\binom{N-1}{m-1}}{\binom{N}{m}} = \frac{m}{N}.$$

The past studies on random graphs were mainly addressed to the type U. However, to handle relations in databases as graph representation, in which tuples tuples (a, b) and (b, a) must be distinguished, and tuples (a, a) may appear, we have to consider directed graphs with self-loops, i.e., the type D_L.

Since we have noticed that we have to use random digraphs to represent relations, we focus only on the random graphs of types D and D_L throughout the rest of this chapter.

5.2.2 Reachability

We give some definitions regarding the relationship between two vertices in a random graph in this subsection.

Definition 5.2:

1. Let u and v be two vertices in a random graph G of type D or D_L.
   - If there is a directed arc from u to v, we say that u is *adjacent* to v, or v is adjacent from u, and denote it by $u \rightarrow v$.
   - Otherwise, we say that u is *not adjacent* to v, or v is not adjacent from u, and denote it by $u \not\rightarrow v$.
2. Let s and t be two vertices in a random graph G of type D or D_L.
   - If there is a directed path from s to t with length more than or equal to 1, we say that s is *reachable* to t, or t is reachable from s, and denote it by $s \rightsquigarrow t$.
   - Otherwise, we say that s is *unreachable* (or *not reachable*) to t, or t is unreachable from s, and denote it by $s \not\rightsquigarrow t$. ■

Example 5.1: In the digraph illustrated in Figure 5.2, s is adjacent to s, t_2 and t_4; s

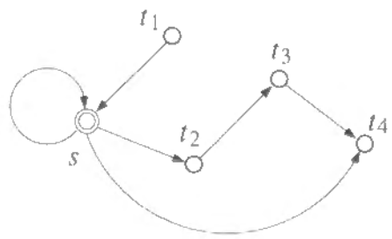


Figure 5.2: A digraph showing reachabilities and adjacencies.

is not adjacent to t_1 and t_3; s is reachable to s, t_2, t_3 and t_4; and s is not reachable to t_1. Conversely, s, t_2 and t_4 are adjacent from s; t_1 and t_3 are not adjacent from s; s, t_2,

t_3 and t_4 are reachable from s; and t_1 is not reachable from s. These relationships are denoted by $s \to s$, $s \to t_2$, $s \to t_4$, $s \not\to t_1$, $s \not\to t_3$, $s \rightsquigarrow s$, $s \rightsquigarrow t_2$, $s \rightsquigarrow t_3$, $s \rightsquigarrow t_4$ and $s \not\rightsquigarrow t_1$. ■

We refer to the arcs appearing in a path from s to t as *forward arcs* from s to t. In a random graph of types D or D_L, we call the probability that s is reachable (or unreachable) to t the *reachability* (or *unreachability*) from s to t. Due to the symmetry of random graphs of the same type, the reachability from s to t ($\neq s$) is independent of the choice of s and t. However, the reachability from s to s (itself) is different from the reachability from s to t ($\neq s$). Thus, we give the following definition.

Definition 5.3: In a random digraph G of types $D(n,p)$ and $D_L(n,p)$, we denote the reachability from s to s and from s to t ($\neq s$) as shown in the following table.

	$s \rightsquigarrow s$	$s \rightsquigarrow t$
G of type $D(n,p)$	$\gamma_{n,p}(s,s)$	$\gamma_{n,p}(s,t)$
G of type $D_L(n,p)$	$\widehat{\gamma}_{n,p}(s,s)$	$\widehat{\gamma}_{n,p}(s,t)$

If there is no confusion and it is obvious from the context in use, we often do not indicate explicitly and omit some of the parameters of the above reachabilities, such as n, p and (s,s) or (s,t). Especially, we often use simply $\gamma_{n,p}$ and $\widehat{\gamma}_{n,p}$ instead of $\gamma_{n,p}(s,t)$ and $\widehat{\gamma}_{n,p}(s,t)$, respectively, for notational convenience. ■

By the definition of reachabilities, we immediately have the following property.

Property 5.4: For the reachabilities $\gamma_{n,p}(s,t)$ and $\widehat{\gamma}_{n,p}(s,t)$,

$$\gamma_{n,p}(s,t) = \widehat{\gamma}_{n,p}(s,t)$$

holds. ■

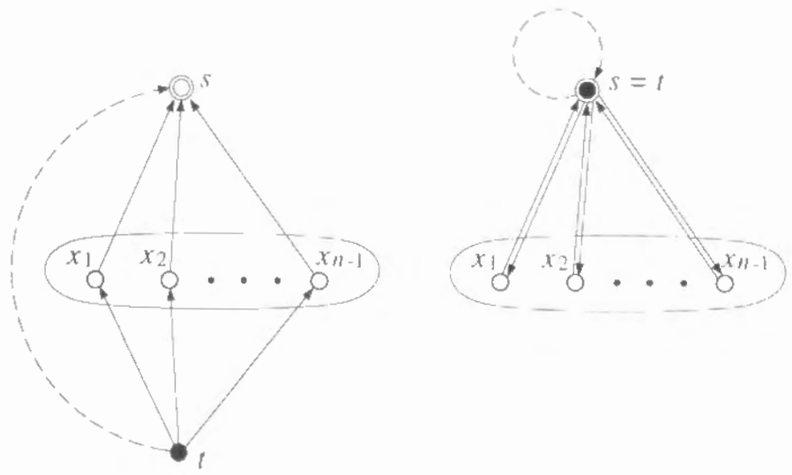
This property is obvious because the reachability from s to t ($\neq s$) is irrelevant to whether there exist self-loops $x_i \to x_i$ or not. From this property, we often do not distinguish between $\gamma_{n,p}(s,t)$ and $\widehat{\gamma}_{n,p}(s,t)$, and denote them $\gamma_{n,p}(s,t)$ or $\gamma_{n,p}$ for short.

Lemma 5.5: For the reachability $\widehat{\gamma}_{n,p}(s,s)$ and $\gamma_{n+1,p}(s,t)$,

$$\widehat{\gamma}_{n,p}(s,s) = \gamma_{n+1,p}(s,t)$$

holds. ■

Proof: Let us consider $\gamma_{n+1,p}(s,t)$, that is, the reachability from s to t in G of type $D(n+1,p)$. In considering the existence of paths from s to t, the arcs $t \to x_i$, $x_i \to s$



(a) Irrelevant arcs in $D(n+1,p)$ (b) New digraph $D_L(n,p)$

Figure 5.3: Proof of Lemma 5.5.

(for any i) and $t \to s$ (shown in Figure 5.3 (a)) are irrelevant. Thus, by deleting these irrelevant arcs and by contracting t and s into one vertex, we can transform the original random digraph G of type $D(n+1,p)$ into the new random digraph G of type $D_L(n,p)$ as shown in Figure 5.3 (b). Furthermore, adding self-loops $x_i \to x_i$ for all i to the digraph does not change the reachability from s to t ($\neq s$). Notice that the arc $s \to t$ with probability p in G of type $D(n+1,p)$ now becomes the self-loop $s \to s$ with probability p in G of type $D_L(n,p)$. All the arcs among vertices $x_1, \ldots, x_{n-1}$ are preserved in the

same manner as G in $D(n+1,p)$. Thus, the resulting digraph now is G of type $D_L(n,p)$. Therefore, the reachability $\gamma_{n+1,p}(s,t)$ is equal to the reachability from s to t $(=s)$ in G of type $D_L(n,p)$, that is, $\widehat{\gamma}_{n,p}(s,s)$. ■

5.2.3 Random Graph and Transitive Closure

We discuss in this subsection, the relationship between the reachability and the expected size of the transitive closure in a random digraph G of type $D_L(n,p)$, which is the average number of pairs of vertices (s,t) such that $s \rightsquigarrow t$ in all the 2^N possible events of G as type D_L. More precisely, the reachabilities $\widehat{\gamma}_{n,p}(s,s)$ and $\widehat{\gamma}_{n,p}(s,t)$ can compute the expected size of the transitive closure from a vertex s, whose definition is given in Definition 2.4. The expected size of the transitive closure from one vertex then gives us the expected size of the transitive closure g_L^+ of a random digraph G of type D_L.

We now introduce the following notations.

Definition 5.6: In a random digraph G of type D_L, we denote the expected number of reachable vertices s from s by $R_L(s,s)$ and the expected number of reachable vertices t $(\neq s)$ from s by $R_L(s,\overline{s})$. Also the expected number of reachable vertices (including s) from a vertex s is denoted by $R_L(s)$. Obviously,

$$R_L(s) = R_L(s,s) + R_L(s,\overline{s})$$

holds by definition. ■

Lemma 5.7: The expected size of the transitive closure of a random digraph G of type $D_L(n,p)$, denoted by $|g_L^+(n,p)|$ (or $|g_L^+|$), is computed by

$$|g_L^+(n,p)| = n(n-1)\gamma_{n,p}(s,t) + n\gamma_{n+1,p}(s,t).$$

■

Proof: The reachability from s to a given vertex t $(\neq s)$ in a random digraph G of type $D_L(n,p)$ is $\widehat{\gamma}_{n,p}(s,t)$, and the expected number of reachable vertices t $(\neq s)$ from s is

$$R_L(s,\overline{s}) = (n-1)\widehat{\gamma}_{n,p}(s,t).$$

Furthermore, the expected number of reachable vertices from s to s is the reachability $\widehat{\gamma}_{n,p}(s,s)$ itself, that is,

$$R_L(s,s) = \widehat{\gamma}_{n,p}(s,s).$$

Therefore, the expected number $R_L(s)$ of the transitive closure of a vertex s is,

$$\begin{aligned} R_L(s) &= R_L(s,\overline{s}) + R_L(s,s) \\ &= (n-1)\widehat{\gamma}_{n,p}(s,t) + \widehat{\gamma}_{n,p}(s,s) \\ &= (n-1)\gamma_{n,p}(s,t) + \gamma_{n+1,p}(s,t) \end{aligned} \tag{5.1}$$

according to Property 5.4 and Lemma 5.5.

Finally, as formula (5.1) holds for any vertex s in a random digraph G of type $D_L(n,p)$, the expected size of the transitive closure of a random digraph G of type $D_L(n,p)$ becomes

$$\begin{aligned} |g_L^+(n,p)| &= nR_L(s) \\ &= n(n-1)\gamma_{n,p} + n\gamma_{n+1,p}, \end{aligned}$$

where we use the simplified notations $\gamma_{n,p}$ and $\gamma_{n+1,p}$ in place of $\gamma_{n,p}(s,t)$ and $\gamma_{n+1,p}(s,t)$. Thus, the proof is completed. ■

This lemma asserts that the expected size of transitive closure of a random digraph G of type $D_L(n,p)$ can be computed if we know the reachability $\gamma_{n,p}$ and $\gamma_{n+1,p}$ in a random digraph $D(n,p)$. Therefore, we have only to concentrate in the subsequent sections on how to compute the reachability $\gamma_{n,p}$ in a random digraph $D(n,p)$. We introduce two methods in the following sections to compute $\gamma_{n,p}$ exactly. Those methods can eventually give us the expected size of the transitive closure of a random digraph of type D_L.

5.3 Computing the Size of Transitive Closure: 1

In this section, we describe the first approach to compute the reachability $\gamma_{n,p}$ ($\gamma_{n,p}(s,t)$), which is defined in a random digraph of type $D(n,p)$.

5.3.1 Basic Idea

This method computes the expected number of reachable vertices ($\neq s$), denoted by $R(s,\overline{s})$, from a source vertex s in a random digraph G of type $D(n,p)$. It counts the number of vertices reachable from s by shortest paths of length i and sums up their numbers from $i=1$ to $n-1$.

For this, let S_i be the set of vertices reachable from s by shortest paths of length i, where $S_0=\{s\}$. Also let

$$T_i = V - \bigcup_{j=0}^{i} S_j,$$

i.e., no vertex in T_i is reachable by a shortest path of length i or less. Furthermore, we denote the number of vertices in S_i by $|S_i|$, for which the following conditions obviously hold:

1. $0 \le |S_i| \le n-1$ for all i,
2. $\sum_{i=1}^{n-1} |S_i| \le n$,
3. If $|S_{i_1}|=0$, then $|S_{i_2}|=0$ for all $i_2 > i_1$.

Now, for a given vector

$$\lambda = (k_1, \ldots, k_{n-1}),$$

we say that an instance of random digraphs G belongs to event λ if $|S_i| = k_i$, $i = 1, \ldots, n-1$, where ($|S_i| =$) k_i's satisfy the above three conditions.

Let us see the meaning of these notations by an example.

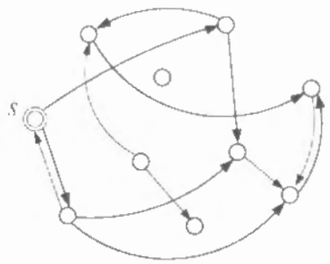


Figure 5.4: An instance of a random digraph G of type $D(10,p)$.

Example 5.2: Consider a random digraph G of type $D(10,p)$, an instance of which is shown in Figure 5.4. Choosing a vertex as a source s, we determine its reachable vertices. Figure 5.5 illustrates the vertices reachable from s by shortest paths, forward arcs and sets S_i and T_i. The random digraph in Figure 5.4 is one of the instances classified as

$$\lambda = (2,3,1,0,0,0,0,0,0).$$

■

Now, if we can compute the occurrence probability of an event $\lambda = (k_1, \ldots, k_{n-1})$, denoted by $P(\lambda)$, we can give the expected number $R(s,\overline{s})$ of vertices reachable from s as

$$R(s,\overline{s}) = \sum_{\lambda} \left(P(\lambda) \sum_{i=1}^{n-1} k_i \right).$$

If we denote the conditional probability of $|S_i| = k_i$ under the condition that ($|S_0| = k_0 = 1$,) $|S_1| = k_1$, ..., $|S_{i-1}| = k_{i-1}$ by $P_i(\lambda)$, the above $P(\lambda)$ can be expressed as

$$P(\lambda) = \prod_{i=1}^{n-1} P_i(\lambda).$$

We then explain that the conditional probability $P_i(\lambda)$ for a given $\lambda = (k_1, \ldots, k_{n-1})$ is determined by k_{i-1}, k_i and $n - \sum_{j=0}^{i-1} k_j$. If $|S_j| = k_j$, $j = 1, \ldots, i-1$, then $|T_{i-1}| = n - \sum_{j=0}^{i-1} k_j$ (i.e., the number of vertices unreachable from s by shortest paths of length $i-1$

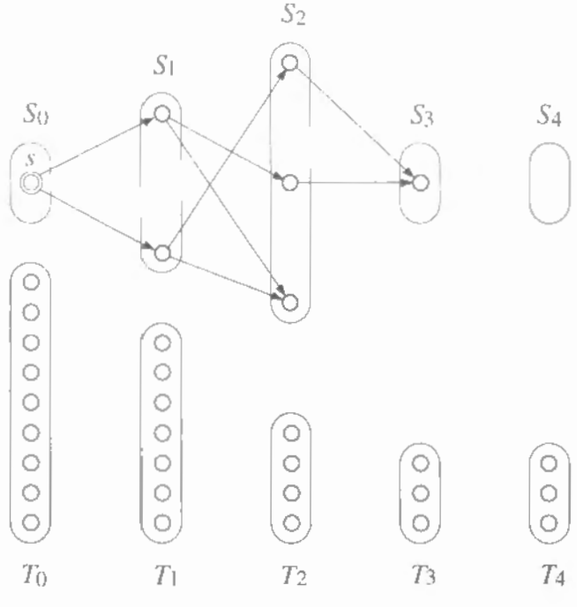


Figure 5.5: Reachable vertices by shortest paths.

or less). First, the probability that a vertex t in T_{i-1} is reachable from no vertex in S_{i-1} is $(1-p)^{k_{i-1}}$, and hence the probability that a vertex t in T_{i-1} is reachable from at least one vertex in S_{i-1} is $1-(1-p)^{k_{i-1}}$. Therefore, the probability that $k_i = |S_i|$ vertices in T_{i-1} is reachable from S_{i-1}, which we denote by $P_i(\lambda)$, becomes

$$P_i(\lambda) = \begin{cases} \binom{|T_{i-1}|}{k_i} \left\{1-(1-p)^{k_{i-1}}\right\}^{k_i} \left\{(1-p)^{k_{i-1}}\right\}^{|T_{i-1}|-k_i} & (k_{i-1} > 0), \\ 1 & (k_{i-1} = 0). \end{cases}$$

Finally, we can compute the reachability $\gamma_{n,p}$ by

$$\begin{aligned} \gamma_{n,p} &= R(s,\overline{s})/(n-1) \\ &= \sum_{\lambda} \left\{ \prod_{i=1}^{n-1} P_i(\lambda) \sum_{i=1}^{n-1} k_i \right\} \Big/ (n-1). \end{aligned}$$

Example 5.3: For an event $\lambda = (2,3,1,0,0,0.0,0,0)$ of Example 5.2, we compute $P(\lambda) = \prod_{i=1}^{9} P_i(\lambda)$. The sizes k_i and $|T_i|$ are shown below,

i	0	1	2	3	4	5	6	7	8	9
k_i	1	2	3	1	0	0	0	0	0	0
$\lvert T_i\rvert$	9	7	4	3	3	3	3	3	3	3

and we can compute

$$\begin{aligned} P_1(\lambda) &= \binom{|T_0|}{k_1} \left\{1-(1-p)^{k_0}\right\}^{k_1} \left\{(1-p)^{k_0}\right\}^{|T_0|-k_1} \\ &= \binom{9}{2} \left\{1-(1-p)^1\right\}^2 \left\{(1-p)^1\right\}^7. \end{aligned}$$

Similarly, we have

$$P_2(\lambda) = \binom{7}{3} \left\{1-(1-p)^2\right\}^3 \left\{(1-p)^2\right\}^4,$$

$$P_3(\lambda) = \binom{4}{1} \left\{1-(1-p)^3\right\}^1 \left\{(1-p)^3\right\}^3,$$

$$P_3(\lambda) = \binom{3}{0} \left\{1-(1-p)^1\right\}^0 \left\{(1-p)^1\right\}^3,$$

and

$$P_5(\lambda) = \cdots = P_9(\lambda) = 1 \quad (\text{since } k_4 = \cdots = k_8 = 0).$$

Therefore,

$$P(\lambda) = \binom{9}{2}(1-q^1)^2(q^1)^7 \binom{7}{3}(1-q^2)^3(q^2)^4 \binom{4}{1}(1-q^3)^1(q^3)^3 \binom{3}{0}(1-q^1)^0(q^1)^3,$$

where $q \equiv 1-p$. ■

5.3.2 An Algorithm for Computing the Exact Reachability

The idea in the previous subsection is easily implemented as an algorithm as shown in Program 5.1, where $Q(i,\lambda)$ indicates the conditional probability $P_i(\lambda)$ in the above

```
Procedure Strict-Reachability0
Input: A random graph G of type D(n,p)
Output: Reachability γ_{n,p}
1    begin
2    for all possible vector λ = (k(1),...,k(n-1)) do
3      P(λ) = 1; S = 1;
4      begin
5      for i := 1 to n-1 do
6        begin
7          P(λ) = P(λ) * Q(i,λ);
8          S = S + k(i);
9        end
10     s+ = s+ + P(λ) * S;
11     end
12   γ_{n,p} = s+/(n-1);
13   end.
```

Program 5.1: Procedure for computing reachability.

explanations. As explained in the previous subsection, the conditional probability $P_i(\lambda)$ is determined only by k_{i-1}, k_i and $|T_{i-1}|$. In other words, since $P_i(\lambda)$ depends only on k_{i-1}, k_i and $\sum_{j=0}^{i-1} k_j$ but not on all sizes k_j $(j = 0, \ldots, i-1)$, we can combine the λ's for which k_{i-1}, k_i and $\sum_{j=0}^{i-1} k_j$ are the same. This saves the computational time of the above initial procedure including the fundamental idea.

In order to implement this idea as a procedure, we introduce the following notations: $S(i)$ and $T(i)$ imply the sizes of sets S_i and T_i, respectively; $P(i,j,k)$ is the conditional probability that $|S_i| = j$ and the sum of $S_1, \ldots, S_i$ is k in the i-th iteration; $con(p,k)$ is the function that computes the probability that any vertex is reachable from at least one of k vertices when the probability of the presence of an arc between any two vertices is p; $bin(m,l,p)$ is the function that computes the probability of binary distribution; s^+

is the size of the transitive closure of single vertex s. The resulting procedure which is implemented with this improvement is shown in Program 5.2.

Theorem 5.8: The procedure Strict-Reachability1 computes the exact reachability $\gamma_{n,p}$ (s,t) in a random digraph G of type $D(n,p)$. This procedure works in $O(n^4)$ time, where n is the number of vertices in a random digraph. ■

We illustrate in Figure 5.6 the reachabilities $\gamma_{n,p}$ with respect to the initial probability p for the presence of an arc between any two vertices, and in Figure 5.7 the reachabilities $\gamma_{n,p}$ with respect to n. Since this procedure requires much computational time, we propose below two kinds of simplifications to compute approximate values of the reachability $\gamma_{n,p}$.

Simplification 1

This idea firstly computes the expected size of S_1 (first iteration) in the same manner as computing S_1 by the procedure Strict-Reachability1. Then, in the second and after iterations, this procedure lets all the sizes of T_i for different $k_i = |S_i|$ be the same value $n - \sum_{j=1}^{i} |S_i|$ irrelevant to the value k_i. In computing $|S_{i+1}|$, it considers $k_{i+1} = 0, \ldots, \lfloor |S_{i+1}| \rfloor$ as the occurrence of S_{i+1}, and it still uses the same value $1-(1-p)^{k_i}$ as in the procedure Strict-Reachability1 for the probability that a vertex t in T_i is reachable from at least one vertex in S_i.

This idea is implemented as in Program 5.3, and it runs $O(n^2)$ time.

Simplification 2

This idea is more simplified and faster than the procedure Simplified-Reachability1, by using the expected sizes for S_1 and T_1 and by using all the expected sizes $S_1, \ldots, S_i, T_1, \ldots, T_i$ for computing S_{i+1}. In this time, the value $1-(1-p)^{|S_i|}$ is used for the probability that a vertex t in T_i is reachable from at least one vertex in S_i. This procedure is shown in

```
Procedure Strict-Reachability1
Input: A random graph G of type D(n,p)
Output: Reachability γ_{n,p}
1     begin
2     P(0,1,1) := 1;
3     for i := 1 to n − 1 do
4       begin
5       for j := i to n − 1 do
6         begin
7         for k := 0 to n − j do
8           begin
9           for ℓ := 0 to n − j − k do
10            begin
11              P(i, j + k, ℓ) := P(i, j + k, ℓ)
12              +P(i − 1, j, k) ∗ bin(n − j − k, ℓ, con(p, k));
13            end
14            S(i) := S(i) + P(i, j + k, ℓ) ∗ ℓ;
15          end
16        end
17        s+ = s+ + S(i);
18      end
19      γ_{n,p} := s+/(n − 1);
20    end.

function con(p, k)
21    return(1 − (1 − p)^k);
function bin(m, l, p)
22    return( (m choose ℓ) p^ℓ (1 − p)^{m−ℓ} );
```

Program 5.2: Procedure for computing the exact reachability.

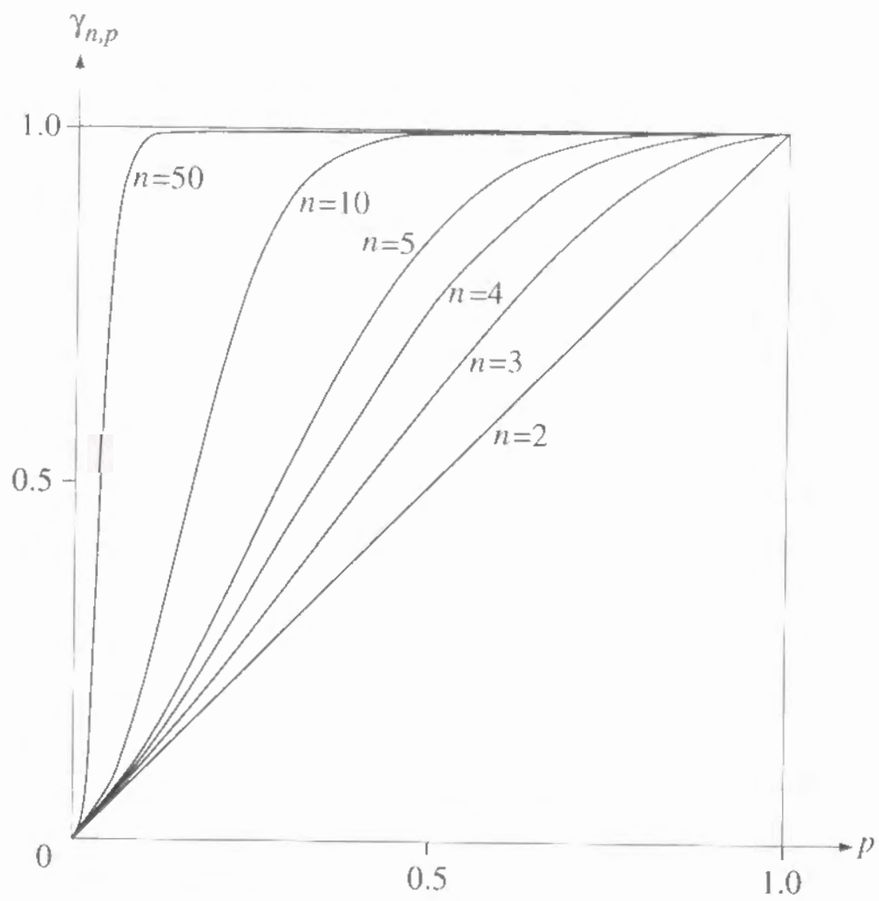


Figure 5.6: Exact reachability $\gamma_{n,p}$ with the initial probability p.

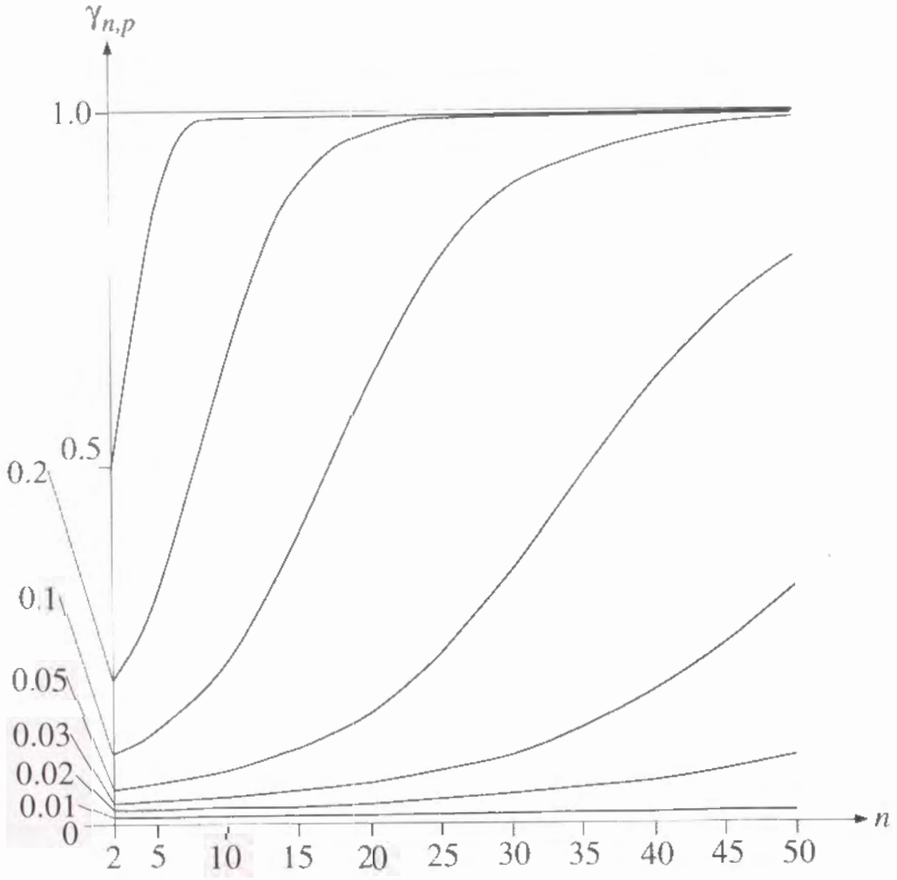


Figure 5.7: Exact reachability $\gamma_{n,p}$ with the number of vertices n.

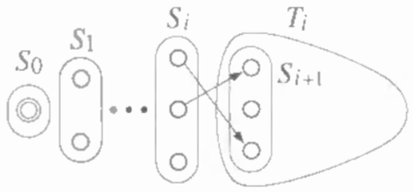

```
Procedure Simplified-Reachability1
Input: A random graph G of type D(n,p)
Output: Approximate Reachability of γ_{n,p}
1     begin
2     S(0) = 1, s+ = 1;
3     for i := 1 to n − 1 do
4       begin
5       for ℓ := 0 to ⌊n − S(i)⌋ do
6         begin
7           S(i) = S(i) + bin(n − s+, ℓ, con(p, s+)) ∗ ℓ;
8         end
9         s+ = s+ + S(i);
10      end
11      γ_{n,p} = s+/(n − 1);
12    end.

function con(p, k)
13      return(1 − (1 − p)^k);
function bin(m, l, p)
14      return( (m choose ℓ) p^ℓ (1 − p)^{m−ℓ} );
```

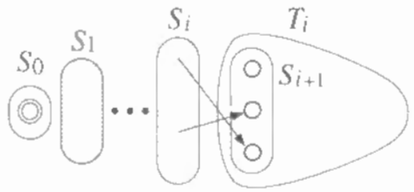
Program 5.3: Simplified procedure 1 for computing reachability.

Program 5.4, and the computational complexity of this procedure is $O(n)$. Furthermore, the image of the difference among the ideas of these procedures is illustrated in Figure 5.8.

Now, we show the computational results of these two simplified procedures in Figure 5.9.

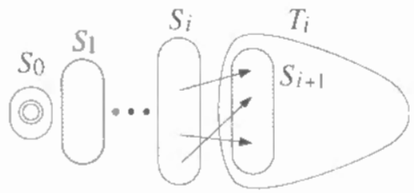


(a) Strict-Reachability1



(b) Simplified-Reachability1

: Expected Values



(c) Simplified-Reachability2

Figure 5.8: Illustration of the three procedures.

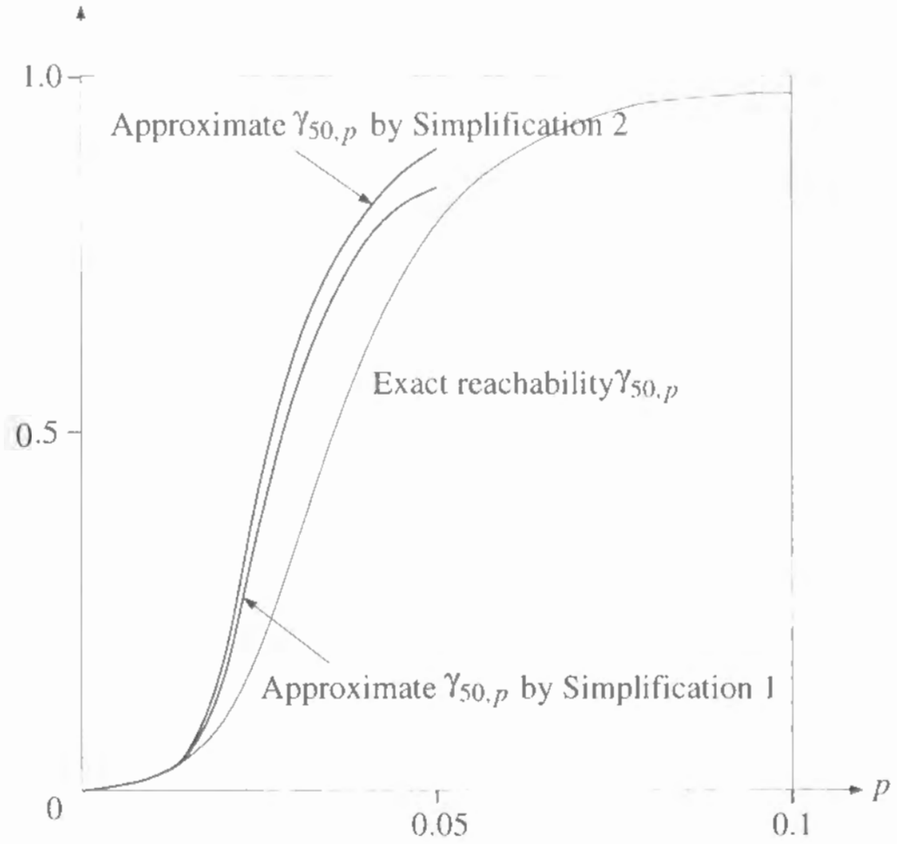


Figure 5.9: Performance of simplified procedures.

```
Procedure Simplified-Reachability2
Input: A random graph G of type D(n,p)
Output: Approximate Reachability of γ_{n,p}
1   begin
2   S(0) = 1, T(0) = n − 1, s⁺ = 1;
3   for i := 1 to n − 1 do
4     begin
5       S(i) = {1 − (1 − p)^{S(i−1)}} ∗ T(i − 1);
6       T(i) = T(i − 1) − S(i);
7       s⁺ = s⁺ + S(i);
8     end
9   γ_{n,p} = s⁺/(n − 1);
10  end.
```

Program 5.4: Simplified procedure 2 for computing reachability.

5.4 Computing the Size of Transitive Closure: 2

In this section, we introduce another method of computing the reachability $\gamma_{n,p}(s,t)$ in a random digraph G of type $D(n,p)$. It begins with $\gamma_{2,p}$ and compute $\gamma_{i,p}$ from $\gamma_{i-1,p}$ for $i = 3, \ldots$ until it finally computes $\gamma_{n,p}$.

We first introduce the definition of the conditional reachability.

Definition 5.9: In a random digraph G of type $D(n,p)$, we denote the restricted reachability from a source vertex s to a sink vertex t under the condition that there is a specified set of j vertices (other than s and t) which are known not to be reachable from s by $\gamma_{n,p}^{(-j)}(s,t)$. Especially, we set

$$\begin{cases} \gamma_{n,p}^{(-0)}(s,t) = \gamma_{n,p}(s,t), \\ \gamma_{n,p}^{(-j)}(s,t) = 0 \quad (j < 0), \end{cases}$$

for convenience. ■

We often use $\gamma_{n,p}^{(-j)}$ instead of $\gamma_{n,p}^{(-j)}(s,t)$, similarly to the case of $\gamma_{n,p}$, for notational convenience when s and t are obvious from the context.

Lemma 5.10: The reachability $\gamma_{n,p}$ in a random digraph G of type $D(n,p)$ is given by the following formula:

$$\gamma_{n,p} = 1 - (1-p)\sum_{i=0}^{n-2}\left[\binom{n-2}{i}(1-p)^{n-2-i}p^i\prod_{j=0}^{i}(1-\gamma_{n-1,p}^{(-j-1)})\right] \quad (n \geq 3). \quad (5.2)$$

■

Proof: While $\gamma_{n,p}$ implies the reachability from s to t, $1-\gamma_{n,p}$ is the unreachability from s to t, that is, formula (5.2) is equivalent to

$$1 - \gamma_{n,p} = (1-p)\sum_{i=0}^{n-2}\left[\binom{n-2}{i}(1-p)^{n-2-i}p^i\prod_{j=0}^{i}(1-\gamma_{n-1,p}^{(-j-1)})\right]. \quad (5.3)$$

We explain this formula by using Figure 5.10. First of all, for t not to be reachable from

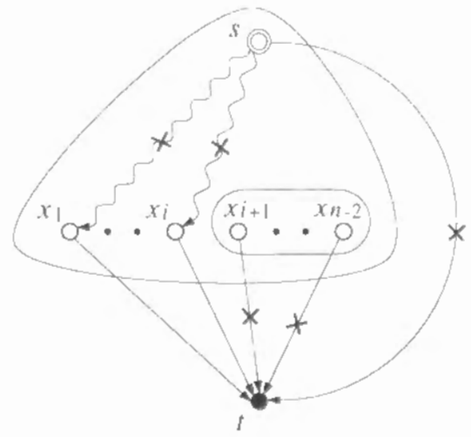


Figure 5.10: Proof of Lemma 5.10.

s, there must not exist an arc $s \to t$ (with probability $1-p$). Then, concerning with all the $n-2$ vertices in $V - \{s,t\}$, let us assume that there are i vertices $x_1, \ldots, x_i$ which

do not have arcs to t; the remaining $n-2-i$ vertices $x_{i+1}, \ldots, x_{n-2}$ do not have arcs to t. The probability of such an event is $\binom{n-2}{i}(1-p)^{n-2-i}p^i$. In this case, all the vertices $x_1, \ldots, x_i$ must not be reachable from s, because otherwise, s would be reachable to t. That is, the vertex x_1 is not reachable from s (with probability $1-\gamma_{n-1,p}$), the vertex x_2 is not reachable from s under the condition that x_1 is not reachable from s (with probability $1-\gamma^{(-1)}_{n-1,p}$), and the final vertex x_i is not reachable from s under the condition that $x_{i+1}, \ldots, x_{n-3}$ are not reachable from s (with probability $1-\gamma^{(-i-1)}_{n-1,p}$). (Here, we can select vertices $x_1, \ldots, x_i$ in this order without loss of generality.) This argument leads to formula (5.3). ■

In order to compute $\gamma^{(-j)}_{n,p}$ in formula (5.2), we have the following lemma.

Lemma 5.11: The conditional reachability $\gamma^{(-j)}_{n,p}$ is computed by

$$\gamma^{(-j)}_{n,p} = \frac{q^{2j}\sum_{i=0}^{n-j-2}\left[\binom{n-j-2}{i}(q^j)^{n-j-2-i}(1-q^j)^i\gamma^{(-i)}_{n-j,p}\prod_{k=0}^{i-1}(1-\gamma^{(-k-1)}_{n-j,p})\right]}{\prod_{k=0}^{j-1}(1-\gamma^{(-k)}_{n,p})}, \quad (5.4)$$

where $q \equiv 1-p$. ■

Before providing the proof of this lemma, we consider its meaning by a simple example.

Example 5.4: Let us consider the conditional reachability $\gamma^{(-1)}_{4,p}$. This is computed as

$$\gamma^{(-1)}_{4,p} = \frac{(1-p)^2\left\{(1-p)\gamma_{3,p} + p(1-\gamma_{3,p})\gamma^{(-1)}_{3,p}\right\}}{1-\gamma_{4,p}} \quad (5.5)$$

by Lemma 5.11. This is the probability of the event

$$\frac{\text{Prob}\ (s \not\rightsquigarrow x_1 \wedge s \rightsquigarrow t)}{\text{Prob}\ (s \not\rightsquigarrow x_1)}$$

in Figure 5.11, where we assume that x_1 is a vertex unreachable from s. The probability that there is no path from s to x_1 ($s \not\rightsquigarrow x_1$) is $1-\gamma_{4,p}$ by definition. Then, to satisfy the joint condition $s \not\rightsquigarrow x_1 \wedge s \rightsquigarrow t$, there must not exist the arc $s \rightarrow x_1$ or $t \rightarrow x_1$ (since otherwise there is a path from s to t). The probability for this is $(1-p)^2$. In addition

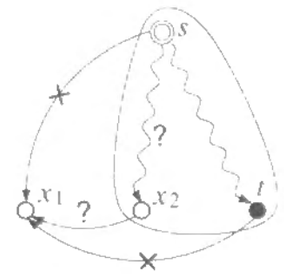


Figure 5.11: The conditional reachability $\gamma^{(-1)}_{4,p}$.

to this condition, we have to consider two cases that there exists an arc from x_2 to x_1 ($x_2 \rightarrow x_1$) and not ($x_2 \not\rightarrow x_1$):

1. If there is no arc from x_2 to x_1 (with probability $1-p$), the condition $s \rightsquigarrow t$ holds only by using some of the three vertices s, t and x_2, and this probability is $\gamma_{3,p}$.
2. If there is an arc from x_2 to x_1 (with probability p), there is no path from s to x_2 by condition $s \not\rightsquigarrow x_2$ (this probability is $1-\gamma_{3,p}$). The probability of $s \rightsquigarrow t$ under the condition $s \not\rightsquigarrow x_2$ is $\gamma^{(-1)}_{3,p}$.

This leads to the conditional probability $\gamma^{(-1)}_{4,p}$ of formula (5.5). ■

Now, we can present the proof of Lemma 5.11.

Proof of Lemma 5.11: We prove this lemma by using Figure 5.12. Let $x_1, \ldots, x_j$ be the vertices unreachable from s. The denominator

$$(1-\gamma_{n,p})(1-\gamma^{(-1)}_{n,p})\cdots(1-\gamma^{(j-1)}_{n,p}) = \prod_{k=0}^{j-1}(1-\gamma^{(-k)}_{n,p}),$$

expresses the probability that j vertices $x_1, \ldots, x_j$ out of $n-2$ vertices are not reachable from s, i.e., this is the product of the unreachabilities $1-\gamma^{(-k)}_{n,p}$ assuming that there are already k unreachable vertices. On the other hand, as explained below, the numerator expresses the reachability under the condition that there are j unreachable vertices. Since

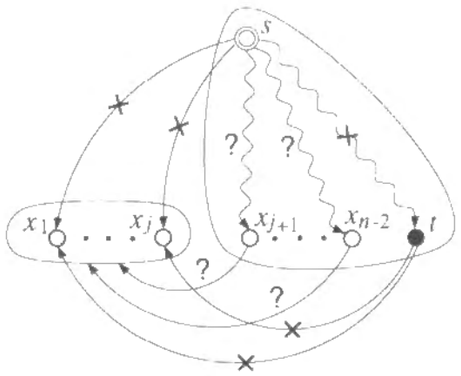


Figure 5.12: Proof of Lemma 5.11.

s is reachable to none of $x_1, \ldots, x_j$, there are no arcs $s \rightarrow x_1, \ldots, s \rightarrow x_j$ or $t \rightarrow x_1, \ldots, t \rightarrow x_j$, because it is assumed that there is a path $s \rightsquigarrow t$. The probability for this condition is q^{2j}. Then, we consider the probability of the existence of a path $s \rightsquigarrow t$ under the condition that i vertices among $n-j-2$ vertices $x_{j+1}, \ldots, x_{n-2}$ are reachable to at least one of the vertices $x_1, \ldots, x_j$. The number of choices of i vertices from $n-j-2$ vertices is $\binom{n-j-2}{i}$ and the probability that exactly these i vertices are reachable to at least one of the j vertices $x_1, \ldots, x_j$ is $(q^j)^{n-j-2-i}(1-q^j)^i$. Then, all of these i vertices must not be reachable from s. This probability that the first vertex is unreachable from s, the second vertex is unreachable from s under the condition that one vertex is unreachable from s, ..., and the i-th (final) vertex is unreachable from s under the condition that $i-1$ vertices are unreachable from s, is $\prod_{k=0}^{i-1}(1-\gamma_{n-j,p}^{(-k-1)})$. Finally, t must be reachable from s by using only $n-j$ vertices s, t, and $x_{j+1}, \ldots, x_{n-2}$ under the condition that the set of i vertices among $x_{j+1}, \ldots, x_{n-2}$ are not reachable from s. This probability is $\gamma_{n-j,p}^{(-i)}$. All of the above probabilities lead to the conditional reachability $\gamma_{n,p}^{(-j)}$ of formula (5.4). ■

Note that, in the formula (5.4) of Lemma 5.11, the conditional probability $\gamma_{n,p}^{(-j)}$ can be computed from reachabilities $\gamma_{n-j,p}^{(-k)}$'s and $\gamma_{n,p}^{(-k)}$'s with $j > 0$ and $k < j$. For example, for $n = 5$, we have

$$\gamma_{5,p}^{(-1)} = \frac{(1-p)^2\left[(1-p)^2\gamma_{4,p} + 2p(1-p)(1-\gamma_{4,p})\gamma_{4,p}^{(-1)} + p^2(1-\gamma_{4,p})(1-\gamma_{4,p}^{(-1)})\gamma_{4,p}^{(-2)}\right]}{(1-\gamma_{5,p})},$$

$$\gamma_{5,p}^{(-2)} = \frac{(1-p)^4\left[(1-p)^2\gamma_{3,p} + \{1-(1-p)^2\}(1-\gamma_{3,p})\gamma_{3,p}^{(-1)}\right]}{(1-\gamma_{5,p})(1-\gamma_{5,p}^{(-1)})},$$

$$\gamma_{5,p}^{(-3)} = \frac{(1-p)^6\gamma_{2,p}}{(1-\gamma_{5,p})(1-\gamma_{5,p}^{(-1)})(1-\gamma_{5,p}^{(-2)})}.$$

Now, Lemma 5.10 and 5.11 establish the procedure for computing the reachability $\gamma_{n,p}$ in Program 5.5, and the next theorem. In this program, $\gamma(k,j)$ denotes the probability $\gamma_{k,p}^{(-j)}$; the outermost loop of lines from 3 to 27 are iterated by k from $k = 3$ to n; lines from 4 to 11 compute $\gamma_{k,p}$, where $\pi_1(i)$ denotes the product $\prod_{j=0}^{i}(1-\gamma_{k-1,p}^{(-j-1)})$ and $q_1(k,0)$ denotes $\sum_{i=0}^{k-2}\left[\binom{k-2}{i}(1-p)^{n-2-i}p^i\prod_{j=0}^{i}(1-\gamma_{k-1,p}^{(-j-1)})\right]$ in formula (5.3); lines from 12 to 26 compute the conditional reachabilities $\gamma_{n-j,p}^{(-j)}$ from $j = 1$ to $k-1$ recursively using the conditional probabilities that are already computed by the $(k-1)$-th iteration, where $\pi_2(i)$ denotes $\gamma_{n-j,p}^{(-i)}\prod_{k=0}^{i-1}(1-\gamma_{n-j,p}^{(-k-1)})$, $\pi_3(i)$ denotes the denominator in formula (5.4), and $q_2(i)$ denotes the numerator in formula (5.4) without being multiplied by q^{2j}.

Theorem 5.12: The procedure Strict-Reachability2 computes the reachability $\gamma_{n,p}$ of a random graph G of type $D(n,p)$. This procedure runs in $O(n^4)$ time. ■

Proof: The correctness of this procedure is obvious from the previous discussions. Since the depth of the loops by n is 4, the time complexity of this procedure is $O(n^4)$. ■

Needless to say, the reachability computed by this procedure is the same as the one computed by the procedure Strict-Reachability1 in Section 5.3.

```
Procedure Strict-Reachability2
Input: A random graph G of type D(n,p)
Output: Reachability γ_{n,p}
 1    begin
 2    γ(2,0) := p;
 3    for k := 3 to n do begin
 4      for i := 0 to k − 2 do begin
 5        π_1(i) := 1; q_1(k,0) := 0;
 6        for j := 0 to i do begin
 7            π_1(i) := π_1(i) * (1 − γ(k − 1, j));
 8          end
 9          q_1(k,0) := q_1(k,0) + C(n−2, k)(1 − p)^{k−2−i} p^i * π_1(i);
10        end
11        γ(k,0) := 1 − (1 − p) * q(k,0);
12        for j := 1 to k − 1 do begin
13          for i := 0 to k − j − 2 do begin
14            π_2(i) := 1; π_3(i) := 1; q_2(k,0) := 0;
15            for ℓ := 0 to i do begin
16                π_2(i) := π_2(i) * (1 − γ(k − j, ℓ − 1));
17              end
18              π_2(i) := π_2(i) * γ(k − j, k − j − 2);
19              q_2(k,0) := q_2(k,0)
                  + C(n−j−2, i)(1 − p)^{j(k−j−2−i)}(1 − (1 − p)^j)^i * π_2(i);
20            for i := 0 to j − 1 do begin
21                π_3(j − 1) := π_3(j − 1) * (1 − γ(k, i));
22              end
23              γ(k, j) := (1 − p)^{2j} * q_2(k, j)/π_3(j − 1);
24            end
25          end
26        end
27      end
28      γ_{n,p} := γ(n,0);
29    end.
```

Program 5.5: Another procedure for computing the exact reachability.

5.5 Lower and Upper Bounds on the Reachability

Since the formulas in Lemmas 5.10 and 5.11 are difficult to handle, we simplify these to obtain upper and lower bounds on the reachability $\gamma_{n,p}$.

5.5.1 Preliminary Lemmas

We firstly present two preliminary lemmas.

Lemma 5.13: Between the reachabilities $\gamma_{n-1,p}$ and $\gamma_{n,p}$, the inequality

$$\gamma_{n-1,p} \leq \gamma_{n,p}$$

holds for $n = 3, 4, \ldots$.

Proof: Consider to add a vertex x to a random graph G of type $D(n-1,p)$ to construct a graph of type $D(n,p)$, as illustrated in Figure 5.13. It is obvious that the inequality

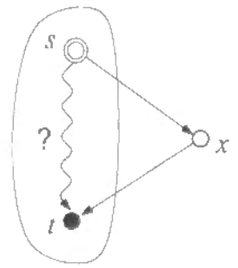


Figure 5.13: Proof of Lemma 5.13.

$$\begin{aligned}\gamma_{n,p} &\geq p^2 + (1-p^2)\gamma_{n-1,p} \\ &= \gamma_{n-1,p} + (1-\gamma_{n-1,p})p^2 \geq \gamma_{n-1,p}\end{aligned}$$

holds for any n (≥ 3). ■

By using this lemma, we can lead a relationship between the reachabilities and the conditional reachabilities.

Lemma 5.14: Between $\gamma_{n,p}^{(-k)}$ and $\gamma_{n-k,p}$, the following inequalities

$$\gamma_{n,p}^{(-k)} \leq \gamma_{n-k,p}$$

hold for $n = 3, 4, \ldots$, and $k = 1, 2, \ldots, n-2$. ■

Proof: In a random graph G of type $D(n,p)$, consider the reachability $\gamma_{n,p}^{(-k)}$ from s to t $(\neq s)$ under the assumption that k vertices $x_1, \ldots, x_k$ are unreachable from s. Let S_i $(i = 1, \ldots, k)$ be the set of vertex x_i and those reachable to x_i, and let $S = \bigcup_i S_i$. This situation is illustrated in Figure 5.14. By assumption, no vertex in each S_i and therefore

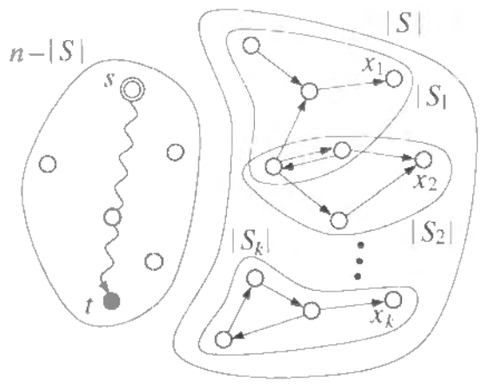


Figure 5.14: Proof of Lemma 5.14.

in S is reachable from s. Now, denote the probability of the occurrence of S by $P(S)$. For this, we have

$$\gamma_{n,p}^{(-k)} = \frac{\sum_{\substack{S=\cup_i S_i \\ S_i \subseteq V-\{s,t\}}} P(S) \cdot \gamma_{n-|S|,p}}{\sum_{\substack{S=\cup_i S_i \\ S_i \subseteq V-\{s,t\}}} P(S)}$$

$$\leq \frac{\sum_{\substack{S=\cup_i S_i \\ S_i \subseteq V-\{s,t\}}} P(S) \cdot \gamma_{n-k,p}}{\sum_{\substack{S=\cup_i S_i \\ S_i \subseteq V-\{s,t\}}} P(S)}$$

$$= \gamma_{n-k,p} \times \frac{\sum_{\substack{S=\cup_i S_i \\ S_i \subseteq V-\{s,t\}}} P(S)}{\sum_{\substack{S=\cup_i S_i \\ S_i \subseteq V-\{s,t\}}} P(S)} = \gamma_{n-k,p},$$

because the set S includes at least k vertices, and therefore $\gamma_{n\ |S|,p} \leq \gamma_{n-k,p}$ holds for all $|S|$ such that $|S| \geq k$. ■

Summarizing the results of Lemmas 5.13 and 5.14, we always have

$$\gamma_{n,p}^{(-k)} \leq \gamma_{n-k,p} \leq \gamma_{n-k+1}$$

for all $n = 3, 4, \ldots$, and $k = 1, 2, \ldots, n-2$.

5.5.2 An Upper Bound on the Reachability

By using the above lemmas and Theorem 5.10, we can obtain a formula for computing an upper bound on the reachability $\gamma_{n,p}$.

Theorem 5.15: The $\gamma_{n,p}^{U}$ obtained by the recursive formula

$$\begin{cases} \gamma_{2,p}^{U} = p, \\ \gamma_{n,p}^{U} = 1 - (1-p)(1 - p \cdot \gamma_{n-1,p}^{U})^{n-2} \quad (n \geq 3) \end{cases}$$

gives an upper bound on the reachability $\gamma_{n,p}$ in a random digraph G of type $D(n,p)$, that is,

$$\gamma_{n,p}^{U} \geq \gamma_{n,p}, \quad n = 2, 3, \ldots$$

holds. ■

Proof: First, $\gamma^U_{2,p}(=p) \geq \gamma_{2,p} = p$ is obvious. For $n \geq 3$, according to Lemma 5.10 and Lemma 5.13,

$$\begin{aligned}
1-\gamma_{n,p} &= (1-p)\sum_{i=0}^{n-2}\left[\binom{n-2}{i}(1-p)^{n-2-i}p^i\prod_{j=0}^{i}\left(1-\gamma^{(-j-1)}_{n-1,p}\right)\right] \\
&\geq (1-p)\sum_{i=0}^{n-2}\left[\binom{n-2}{i}(1-p)^{n-2-i}p^i(1-\gamma_{n-1,p})^i\right] \\
&= (1-p)\Big\{(1-p)+p(1-\gamma_{n-1,p})\Big\}^{n-2} \\
&= (1-p)(1-p\cdot\gamma_{n-1,p})^{n-2}.
\end{aligned}$$

Therefore, for any upper bound $\gamma^U_{n-1,p}$ on $\gamma_{n-1,p}$, we have

$$(1-p\cdot\gamma^U_{n-1,p})^{n-2} \leq (1-p\cdot\gamma_{n,p})^{n-2}$$

and hence

$$(1-p)(1-p\cdot\gamma^U_{n-1,p})^{n-2} \leq (1-p)(1-p\cdot\gamma_{n,p})^{n-2}.$$

Then we have

$$\begin{aligned}
\gamma^U_{n,p} &\equiv 1-(1-p)(1-p\cdot\gamma^U_{n-1,p})^{n-2} \\
&\geq 1-(1-p)(1-p\cdot\gamma_{n-1,p})^{n-2} \\
&\geq \gamma_{n,p},
\end{aligned}$$

namely, $\gamma^U_{n,p}$ in the theorem statement is an upper bound on $\gamma_{n,p}$. ■

5.5.3 A Lower Bound on the Reachability

Theorem 5.16: The $\gamma^L_{n,p}$ obtained by the following recursive formula

$$\begin{cases}
\gamma^L_{2,p} = p, \\
\gamma^L_{n,p} = 1-(1-p)\left\{(1-\gamma^L_{n-1,p})+\gamma^L_{n-1,p}(1-p)^{n-2}\right\} \quad (n\geq 3)
\end{cases}$$

gives a lower bound on the exact reachability $\gamma_{n,p}$ in a random digraph G of type $D(n,p)$, that is,

$$\gamma^L_{n,p} \leq \gamma_{n,p}, \quad n = 2,3,\ldots$$

holds. ■

Proof: First, $\gamma^L_{2,p}(=p) \leq \gamma_{2,p} = p$ is obvious. For $n \geq 3$, according to Theorem 5.10 and Lemma 5.13,

$$\begin{aligned}
1-\gamma_{n,p} &= (1-p)\sum_{i=0}^{n-2}\left\{\binom{n-2}{i}(1-p)^{n-2-i}p^i\prod_{j=0}^{i}\left(1-\gamma^{(-j-1)}_{n-1,p}\right)\right\} \\
&\leq (1-p)\left[\sum_{i=0}^{n-2}\left\{\binom{n-2}{i}(1-p)^{n-2-i}p^i(1-\gamma_{n-1,p})\right\}+\gamma_{n-1,p}(1-p)^{n-2}\right] \\
&= (1-p)\left[(1-\gamma_{n-1,p})\sum_{i=0}^{n-2}\left\{\binom{n-2}{i}(1-p)^{n-2-i}p^i\right\}+\gamma_{n-1,p}(1-p)^{n-2}\right] \\
&= (1-p)\Big\{(1-\gamma_{n-1,p})+\gamma_{n-1,p}(1-p)^{n-2}\Big\}.
\end{aligned}$$

Therefore, for any lower bound $\gamma^L_{n-1,p}$ on $\gamma_{n-1,p}$, by using the inequality

$$\gamma^L_{n-1,p} \geq \gamma^L_{n-1,p}(1-p)^{n-2},$$

we have

$$(1-\gamma^L_{n-1,p})+\gamma^L_{n-1,p}(1-p)^{n-2} \geq (1-\gamma_{n-1,p})+\gamma_{n-1,p}(1-p)^{n-2}$$

and so

$$(1-p)\Big\{(1-\gamma^L_{n-1,p})+\gamma^L_{n-1,p}(1-p)^{n-2}\Big\} \geq (1-p)\Big\{(1-\gamma_{n-1,p})+\gamma_{n-1,p}(1-p)^{n-2}\Big\}.$$

Therefore,

$$\begin{aligned}
\gamma^L_{n,p} &\equiv 1-(1-p)\Big\{(1-\gamma^L_{n-1,p})+\gamma^L_{n-1,p}(1-p)^{n-2}\Big\} \\
&\leq 1-(1-p)\Big\{(1-\gamma_{n-1,p})+\gamma_{n-1,p}(1-p)^{n-2}\Big\} \\
&\leq \gamma_{n,p}.
\end{aligned}$$

■

We illustrate these upper and lower bounds for $n = 50$ in Figure 5.15.

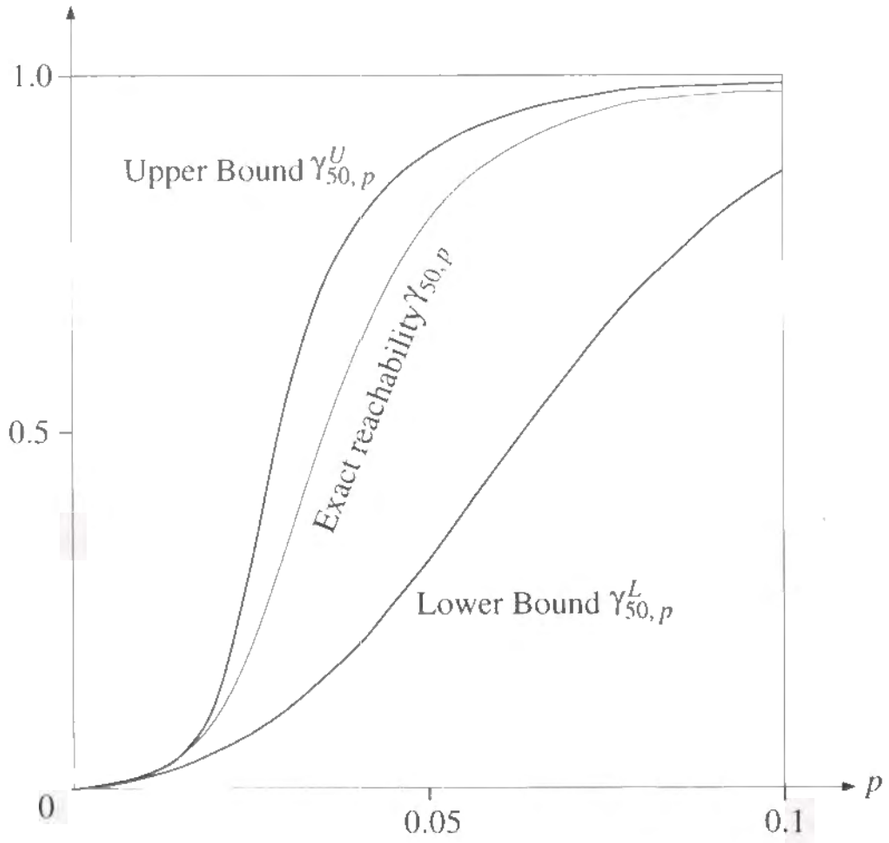


Figure 5.15: The upper and lower bounds on $\gamma_{50,p}$.

5.6 Asymptotic Analysis of the Upper Bound

In this section, we analyze the asymptotic behavior of the upper bound $\gamma^U_{n,p}$ on the reachability $\gamma_{n,p}$ in a random digraph of type $D(n,p)$.

In particular, we are interested in the behavior of the reachability where the initial probability p is given as a function of n, that is p_n. Let us concentrate on the case of

$$p_n = \frac{\alpha}{n-1}, \tag{5.6}$$

where α is a positive constant, that is, a random digraph G of type $D\left(n, \frac{\alpha}{n-1}\right)$. In this case, the upper bound $\gamma^U_{n,\frac{\alpha}{n-1}}$ is computed by the iterative computation in Theorem 5.15,

$$\begin{cases} \gamma^U_{n_0,\frac{\alpha}{n_0-1}} = \dfrac{\alpha}{n-1} \quad (n_0 \text{ satisfies } n_0 \geq \alpha+1), \\ \gamma^U_{i,\frac{\alpha}{n-1}} = 1-\left(1-\dfrac{\alpha}{n-2}\right)\left(1-\dfrac{\alpha}{n-2}\gamma^U_{i-1,\frac{\alpha}{n-1}}\right)^{i-2} \quad (i=n_0+1,\ldots,n), \end{cases} \tag{5.7}$$

where the initial probability p is modified to $p_{n_0} = \dfrac{\alpha}{n_0-1}$ so that it satisfies the condition $0 \leq p_{n_0} \leq 1$.

In order to analyze the behavior of the sequence $\left\{\gamma^U_{n,\frac{\alpha}{n-1}}\right\}$ directly, each $\gamma^U_{n,\frac{\alpha}{n-2}}$ needs to be computed by the $(n-n_0)$-th iteration of the above recursive formula. To simplify this, we introduce another upper bound $\zeta^U_{n,\frac{\alpha}{n-1}}$ on $\gamma^U_{n,\frac{\alpha}{n-1}}$, which will become equal to $\gamma^U_{n,\frac{\alpha}{n-1}}$ as n tends to infinity.

Lemma 5.17: The $\zeta^U_{n,p}$ obtained by the recursive formula

$$\begin{cases} \zeta^U_{n_0,\frac{\alpha}{n_0-1}} = \dfrac{\alpha}{n_0-1} \quad (n_0 \text{ satisfies } n_0 \geq \alpha+1), \\ \zeta^U_{n,\frac{\alpha}{n-1}} = 1-\left(1-\dfrac{\alpha}{n-2}\right)\left(1-\dfrac{\alpha}{n-2}\zeta^U_{n-1,\frac{\alpha}{n-2}}\right)^{n-2} \quad (n>n_0) \end{cases} \tag{5.8}$$

gives an upper bound on $\gamma^U_{n,\frac{\alpha}{n-1}}$ in a random digraph G of type $D\left(n, \frac{\alpha}{n-1}\right)$, that is,

$$\zeta^U_{n,\frac{\alpha}{n-1}} \geq \gamma^U_{n,\frac{\alpha}{n-1}} \quad (n=2,3,\ldots).$$

■

Proof: While $\gamma^U_{n,\frac{\alpha}{n-1}}$ is computed as

$$\gamma^U_{n_0,\frac{\alpha}{n-1}}, \gamma^U_{n_0+1,\frac{\alpha}{n-1}}, \cdots, \gamma^U_{n-1,\frac{\alpha}{n-1}}, \gamma^U_{n,\frac{\alpha}{n-1}},$$

beginning with the initial probability $\gamma^U_{n_0,\frac{\alpha}{n-1}}$, $\zeta^U_{n,\frac{\alpha}{n-1}}$ is computed as

$$\zeta^U_{n_0,\frac{\alpha}{n_0-1}}, \zeta^U_{n_0+1,\frac{\alpha}{(n_0+1)-1}}, \cdots, \zeta^U_{n-1,\frac{\alpha}{(n-1)-1}}, \zeta^U_{n,\frac{\alpha}{n-1}}.$$

In the computation of each $\zeta^U_{n',\frac{\alpha}{n'-1}}$, it is computed by using the final iteration of computing $\gamma^U_{n',\frac{\alpha}{n'-1}}$, that is,

$$\gamma^U_{n',\frac{\alpha}{n'-1}} = 1 - \left(1 - \frac{\alpha}{n'-1}\right)\left(1 - \frac{\alpha}{n'-1}\gamma^U_{n'-1,\frac{\alpha}{n'-1}}\right)^{(n'-1)-1}, \tag{5.9}$$

and substitute $\zeta^U_{n'-1,\frac{\alpha}{(n'-1)-1}}$ in place of $\gamma^U_{n'-1,\frac{\alpha}{n'-1}}$, that is,

$$\zeta^U_{n',\frac{\alpha}{n'-1}} = 1 - \left(1 - \frac{\alpha}{n'-1}\right)\left(1 - \frac{\alpha}{n'-1}\zeta^U_{n'-1,\frac{\alpha}{(n'-1)-1}}\right)^{(n'-1)-1}. \tag{5.10}$$

In formulas (5.9) and (5.10), if $\zeta^U_{n'-1,\frac{\alpha}{(n'-1)-1}} \geq \gamma^U_{n'-1,\frac{\alpha}{n'-1}}$ holds, it is obvious from formulas (5.7) and (5.8) that

$$\left(1-\frac{\alpha}{n'-1}\right)\left(1-\frac{\alpha}{n'-1}\zeta^U_{n'-1,\frac{\alpha}{n'-2}}\right)^{n'-2} \leq \left(1-\frac{\alpha}{n'-1}\right)\left(1-\frac{\alpha}{n'-1}\gamma^U_{n'-1,\frac{\alpha}{n-1}}\right)^{n'-2}$$

holds, and hence

$$\zeta^U_{n',\frac{\alpha}{n'-1}} \geq \gamma^U_{n',\frac{\alpha}{n-1}}.$$

Initially, since we obviously have

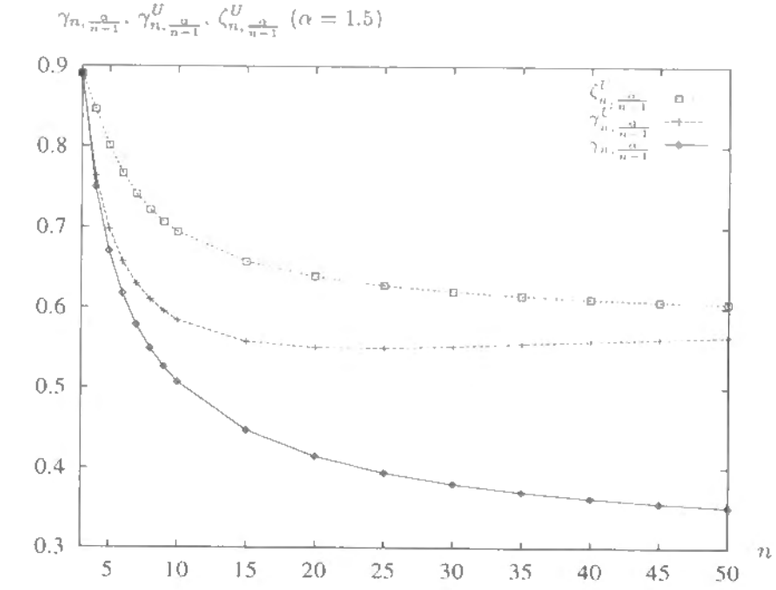
$$\zeta^U_{n_0,\frac{\alpha}{n_0-1}} = \frac{\alpha}{n_0-1} = \gamma^U_{n_0,\frac{\alpha}{n_0-1}} \geq \gamma^U_{n_0,\frac{\alpha}{n-1}},$$

$\zeta^U_{n,\frac{\alpha}{n-1}}$ becomes the upper bound on $\gamma^U_{n,\frac{\alpha}{n-1}}$ by induction of n. ■

We illustrate in Figure 5.16 the behaviors and the relationships among $\gamma_{n,\frac{\alpha}{n-1}}$, $\gamma^U_{n,\frac{\alpha}{n-1}}$ and $\zeta^U_{n,\frac{\alpha}{n-1}}$.

Then, we have the following theorem.

Theorem 5.18: When n tends to infinity, the upper bound ζ^U_{n,p_n} defined by formula (5.8), converges to ζ^U_α, which is the solution x_2 of the equation



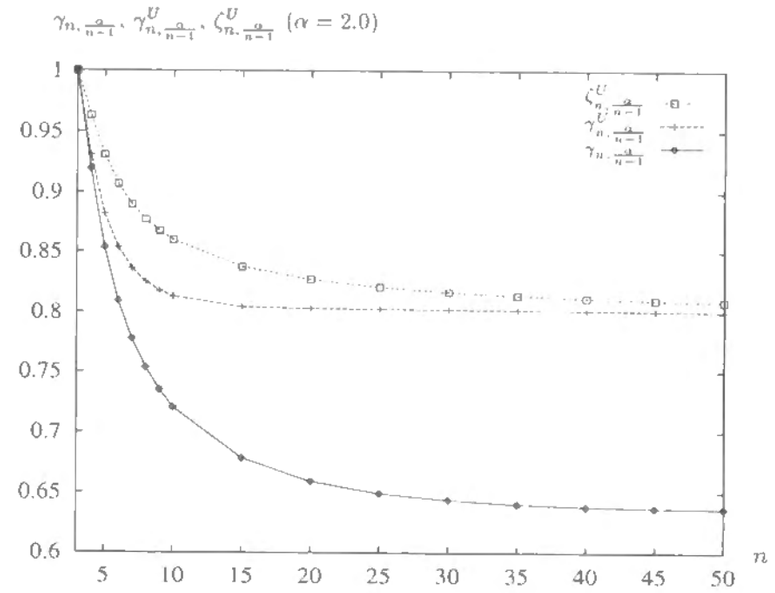


Figure 5.16: Behaviors of $\gamma_{n,\frac{\alpha}{n-1}}$, $\gamma^U_{n,\frac{\alpha}{n-1}}$ and $\zeta^U_{n,\frac{\alpha}{n-1}}$.

$$1 - x - e^{-\alpha x} = 0, \tag{5.11}$$

satisfying $0 < x_2 < 1$ if $\alpha > 1$, and $x_2 = 0$ if $0 < \alpha \leq 1$. (Equation (5.11) has at most two solutions, one which is $x_1 = 0$ and the other solutions x_2 can be positive or negative, depending on α.) ■

Proof: We examine the behavior of ζ^U_{n,p_n} defined by

$$\zeta^U_{n,p_n} = 1 - \left(1 - \frac{\alpha}{n-2}\right)\left(1 - \frac{\alpha}{n-2}\zeta^U_{n-1,p_{n-1}}\right)^{n-2} \quad (n > n_0). \tag{5.12}$$

First of all, each ζ^U_{n,p_n} is bounded as follows:

$$0 \leq \zeta^U_{n,p_n} \leq 1.$$

This can easily be verified by induction of n. Initially, $0 \leq \zeta^U_{n_0,p_{n_0}} = p_{n_0} \leq 1$ is obvious. Suppose that $\zeta^U_{n-1,p_{n-1}}$ is in the range $0 \leq \zeta^U_{n-1,p_{n-1}} \leq 1$. Since $0 \leq \frac{\alpha}{n-1} \leq 1$ holds for $n > n_0$, it holds that $0 \leq \left(1 - \frac{\alpha}{n-1}\zeta^U_{n-1,p_{n-1}}\right) \leq 1$ and $0 \leq \left(1 - \frac{\alpha}{n-1}\right) \leq 1$, and therefore,

$$0 \leq 1 - \left(1 - \frac{\alpha}{n-1}\right)\left(1 - \frac{\alpha}{n-1}\zeta^U_{n,p_n}\right)^{n-1} = \zeta^U_{n+1,p_{n+1}} \leq 1.$$

Next, we show that the sequence $\{\zeta^U_{n,p_n}\}$ becomes monotone for sufficiently large n. Consider the following two consecutive terms:

$$\zeta^U_{n+1,p_{n+1}} = 1 - \left(1 - \frac{\alpha}{n-1}\right)\left(1 - \frac{\alpha}{n-1}\zeta^U_{n,p_n}\right)^{n-1} \equiv 1 - \xi^U_{n+1,p_{n+1}},$$

$$\zeta^U_{n,p_n} = 1 - \left(1 - \frac{\alpha}{n-2}\right)\left(1 - \frac{\alpha}{n-2}\zeta^U_{n-1,p_{n-1}}\right)^{n-2} \equiv 1 - \xi^U_{n,p_n}.$$

First, we show that if $\zeta^U_{n-1,p_{n-1}} \geq \zeta^U_{n,p_n}$, then it implies $\zeta^U_{n,p_n} \geq \zeta^U_{n+1,p_{n+1}}$. To compare the size of two terms $\xi^U_{n+1,p_{n+1}}$ and ξ^U_{n,p_n}, we consider their ratio, that is,

$$\begin{aligned}
&\xi^U_{n+1,p_{n+1}} \Big/ \xi^U_{n,p_n} \\
&= \left(1 - \frac{\alpha}{n-1}\right)\left(1 - \frac{\alpha}{n-1}\zeta^U_{n,p_n}\right)^{n-1} \Big/ \left(1 - \frac{\alpha}{n-2}\right)\left(1 - \frac{\alpha}{n-2}\zeta^U_{n-1,p_{n-1}}\right)^{n-2} \\
&= \frac{n-1-\alpha}{n-1} \cdot \frac{(n-1-\alpha\zeta^U_{n,p_n})^{n-1}}{(n-1)^{n-1}} \cdot \frac{n-2}{n-2-\alpha} \cdot \frac{(n-2)^{n-2}}{(n-2-\alpha\zeta^U_{n-1,p_{n-1}})^{n-2}}
\end{aligned}$$

$$\begin{aligned}
&= \frac{n-1-\alpha}{n-1} \cdot \frac{n-2}{n-2-\alpha} \cdot \frac{n-1-\alpha\zeta^U_{n,p_n}}{n-1} \cdot \frac{(n-1-\alpha\zeta^U_{n,p_n})^{n-2}}{(n-1)^{n-2}} \cdot \frac{(n-2)^{n-2}}{(n-2-\alpha\zeta^U_{n-1,p_{n-1}})^{n-2}} \\
&= \frac{n^2-3n-\alpha n+2\alpha+2}{n^2-3n-\alpha n+\alpha+2} \cdot \frac{n-1-\alpha\zeta^U_{n,p_n}}{n-1} \cdot \left\{\frac{(n-2)(n-1-\alpha\zeta^U_{n,p_n})}{(n-1)(n-1-\alpha\zeta^U_{n-1,p_{n-1}})}\right\}^{n-2}
\end{aligned} \tag{5.13}$$

In formula (5.13), since

$$\begin{aligned}
&\frac{n^2-3n-\alpha n+2\alpha+2}{n^2-3n-\alpha n+\alpha+2} \\
&= 1 + \frac{\alpha}{n^2-3n-\alpha n+\alpha+2} \geq 1,
\end{aligned}$$

and the remaining term

$$\begin{aligned}
&\frac{n-1-\alpha\zeta^U_{n,p_n}}{n-1}\left\{\frac{(n-2)(n-1-\alpha\zeta^U_{n,p_n})}{(n-1)(n-1-\alpha\zeta^U_{n-1,p_{n-1}})}\right\}^{n-2} \\
&= \frac{n-1-\alpha\zeta^U_{n,p_n}}{n-1}\left\{\frac{n^2-3n-\alpha n\zeta^U_{n,p_n}+2+2\alpha\zeta^U_{n,p_n}}{n^2-3n-\alpha n\zeta^U_{n-1,p_{n-1}}+2+\alpha\zeta^U_{n-1,p_{n-1}}}\right\}^{n-2} \\
&= \frac{n-1-\alpha\zeta^U_{n,p_n}}{n-1}\left\{1 + \frac{\alpha\zeta^U_{n,p_n}}{n^2-3n-\alpha n\zeta^U_{n-1,p_{n-1}}+2+\alpha\zeta^U_{n-1,p_{n-1}}}\right\}^{n-2} \\
&\geq \frac{n-1-\alpha\zeta^U_{n,p_n}}{n-1}\left(1 + \frac{(n-2)\alpha\zeta^U_{n,p_n}}{n^2-3n-\alpha n\zeta^U_{n-1,p_{n-1}}+2+\alpha\zeta^U_{n-1,p_{n-1}}}\right) \\
&\geq \frac{n-1-\alpha\zeta^U_{n,p_n}}{n-1}\left(1 + \frac{(n-2)\alpha\zeta^U_{n,p_n}}{n^2-3n-\alpha n\zeta^U_{n,p_n}+2+\alpha\zeta^U_{n,p_n}}\right) \quad (\text{since } \zeta^U_{n,p_n} \geq \zeta^U_{n+1,p_{n+1}}) \\
&= \frac{n-1-\alpha\zeta^U_{n,p_n}}{n-1} \cdot \frac{n^2-3n+2-\alpha\zeta^U_{n,p_n}}{n^2-3n+2-\alpha n\zeta^U_{n,p_n}+\alpha\zeta^U_{n,p_n}} \\
&= \frac{n^3-4n^2+5n-\alpha n^2\zeta^U_{n,p_n}+2\alpha n\zeta^U_{n,p_n}-\alpha\zeta^U_{n,p_n}-2+\alpha^2{\zeta^U_{n,p_n}}^2}{n^3-4n^2+5n-\alpha n^2\zeta^U_{n,p_n}+2\alpha n\zeta^U_{n,p_n}-\alpha\zeta^U_{n,p_n}-2} \\
&= 1 + \frac{\alpha^2{\zeta^U_{n,p_n}}^2}{n^3-4n^2+5n-\alpha n^2\zeta^U_{n,p_n}+2\alpha n\zeta^U_{n,p_n}-\alpha\zeta^U_{n,p_n}-2}
\end{aligned}$$

becomes greater than or equal to 1 (by $\alpha > 0$ and $\zeta^U_{n,p_n} \geq 0$ for $n > n_0$), we can conclude that

$$\frac{\xi^U_{n+1,p_{n+1}}}{\xi^U_{n,p_n}} \geq 1$$

holds for any $n \geq n_0$. This means $\xi^U_{n+2,p_{n+2}} \geq \xi^U_{n+1,p_{n+1}}$ and therefore, we obtain

$$\zeta^U_{n+1,p_{n+1}} \geq \zeta^U_{n+2,p_{n+2}}.$$

This says that once it holds that $\zeta^U_{n'+1,p_{n'+1}} \leq \zeta^U_{n',p_{n'}}$ for any $n' > n_0$, then the sequence $\{\zeta^U_{n,p_n}\}$ becomes non-increasing for all $n > n'$. On the other hand, if no $n' > n_0$ satisfies $\zeta^U_{n'+1,p_{n'+1}} \leq \zeta^U_{n',p_{n'}}$, it means that $\{\zeta^U_{n,p_n}\}$ is increasing. Consequently, the sequence $\{\zeta^U_{n,p_n}\}$ becomes either non-increasing or increasing for sufficiently large $n > n'$.

Since the sequence $\{\zeta^U_{n,p_n}\}$ is bounded from below and from above, the above monotonicity proves that it converges to a certain limit value.

Now, let the limit value of the sequence $\{\zeta^U_{n,p_n}\}$ be ζ^U_α. Then, it has to satisfy the following equation obtained from formula (5.12) by substituting $\zeta^U_{n,p_n} = \zeta^U_{n-1,p_{n-1}} = x$:

$$x = 1 - \left(1 - \frac{\alpha}{n-1}\right)\left(1 - \frac{\alpha}{n-1}x\right)^{n-1}.$$

When n tends to infinity,

$$\left(1 - \frac{\alpha}{n-1}\right) \to 1$$

and

$$\left(1 - \frac{\alpha}{n-1}x\right)^{n-1} \to e^{-\alpha x}.$$

Therefore, $x = \zeta^U_\alpha$ satisfies

$$1 - x - e^{-\alpha x} = 0. \tag{5.14}$$

As shown in Figure 5.17, this equation has two solutions $x_1 = 0$ and x_2 satisfying $0 < x_2 < 1$ if $\alpha > 1$; one multiple solution $x_2 = 0$ if $\alpha = 1$; and two solutions x_1 satisfying $x_1 < 0$ and $x_2 = 0$ if $0 \leq \alpha < 1$. We then assert that

$$\zeta^U_\alpha = x_2.$$

In the case of $0 \leq \alpha \leq 1$, it is obvious that $\zeta^U_\alpha = x_2\ (= 0)$ holds, because ζ^U_α must be in the range $0 \leq \zeta^U_\alpha \leq 1$ and the equation (5.14) has only one solution in that range. On the other hand, in the case of $\alpha > 1$ (i.e., $x_2 > 0$), we consider the following two possible cases:

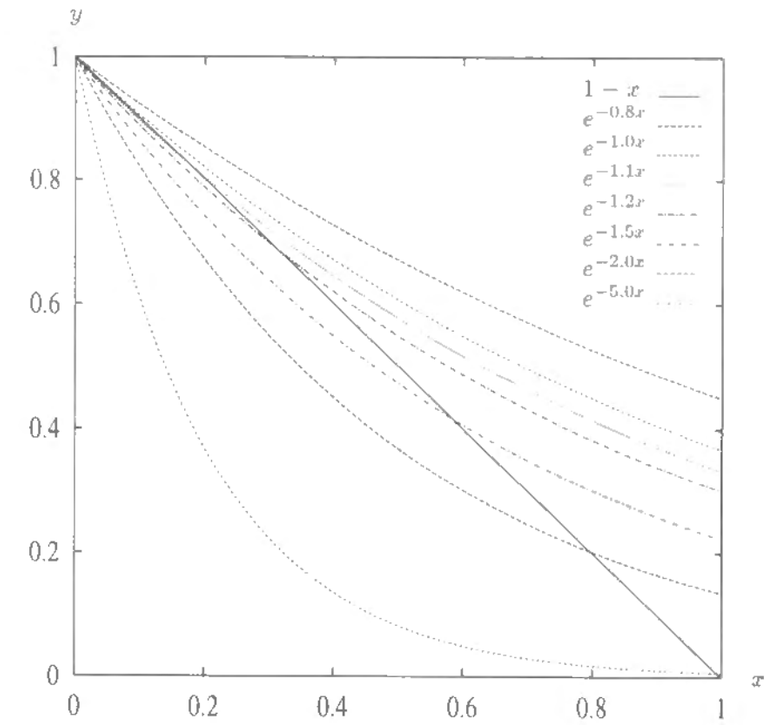


Figure 5.17: Behavior of two functions $1 - x$ and $e^{-\alpha x}$.

1. The sequence $\{\zeta^U_{n,p_n}\}$ is increasing.

   In this case, since the initial probability $\zeta^U_{n_0,p_{n_0}}$ is greater than 0 by $\alpha > 1$, it never converges to 0. Therefore, $\zeta^U_\alpha = x_2$ must hold.

2. The sequence $\{\zeta^U_{n,p_n}\}$ is non-increasing.

   We show that ζ^U_{n,p_n} never be less than $x_2 > 0$ by induction of n. Suppose ζ^U_{n,p_n} $(= t_n)$ is greater than x_2 for some n, that is,

   $$1 - t_n < e^{-\alpha t_n}.$$

   (For example, the initial n_0 satisfies $1 - t_{n_0} < e^{-\alpha t_{n_0}}$ if $\alpha\ (> 1)$ of $t_{n_0} = \frac{\alpha}{n_0 - 1}$

is sufficiently close to 1 because if $n_0 = 3$

$$\lim_{\alpha \to 1} \left(1 - \frac{\alpha}{2}\right) = \frac{1}{2} < \lim_{\alpha \to 1} \left(e^{-\alpha^2/2}\right) = 0.60653 \cdots,$$

holds.) Then, the next $\zeta^U_{n+1,p_{n+1}}$ $(= t_{n+1})$ satisfies

$$t_{n+1} = 1 - \left(1 - \frac{\alpha}{n-1}\right)\left(1 - \frac{\alpha}{n-1} t_n\right)^{n-1}.$$

Since $t_n \geq t_{n+1}$ by the non-increasingness of $\{\zeta^U_{n,p_n}\}$, it holds that

$$1 - t_{n+1} = \left(1 - \frac{\alpha}{n-1}\right)\left(1 - \frac{\alpha}{n-1} t_n\right)^{n-1} < e^{-\alpha t_n} \leq e^{-\alpha t_{n+1}},$$

because $\left(1 - \frac{\alpha}{n-1}\right)$ and $\left(1 - \frac{\alpha}{n-1} t_1\right)^{n-1}$ are monotone increasing in n when $\alpha > 1$, and the latter converges to $e^{-\alpha t_1}$. This implies that $t_n = \zeta^U_{n,p_n}$ never becomes less than x_2, which is a solution of the equation (5.14).

This concludes the proof of this theorem. ■

Now, we may comment upon a relationship between two upper bounds γ^U_{n,p_n} and ζ^U_{n,p_n}. Although we have not proved the convergence of the sequence $\{\gamma^U_{n,p_n}\}$ yet, we can present the following property.

Property 5.19: If the sequence $\{\gamma^U_{n,p_n}\}$ converges as n tends to infinity, it converges to the same solution x_2 of equation (5.14). ■

Proof: We have already proved that ζ^U_{n,p_n}, which is computed by

$$\zeta^U_{n,p_n} = 1 - (1 - p_{n-1})\,(1 - p_{n-1} \cdot \zeta^U_{n-1,p_{n-1}})^{n-2},$$

becomes an upper bound on γ^U_{n,p_n} and converges to x_2. Also note that the final iteration (5.7) of computing γ^U_{n,p_n} for $i = n$ has a similar form:

$$\gamma^U_{n,p_n} = 1 - \left(1 - \frac{\alpha}{n-2}\right)\left(1 - \frac{\alpha}{n-2}\gamma^U_{n-1,p_n}\right)^{n-2}. \tag{5.15}$$

Therefore, we have only to show that $\gamma^U_{n-1,p_n} = \gamma^U_{n,p_n}$ holds in the limit state. Now, since the convergence of sequence $\{\gamma^U_{n,p_n}\}$ implies that $\gamma^U_{n-1,p_{n-1}} = \gamma^U_{n,p_n}$, it suffices to show

$\gamma^U_{n-1,p_{n-1}} = \gamma^U_{n-1,p_n}$ in the limit state, that is, the ratio $\gamma^U_{n-1,p_n}/\gamma^U_{n-1,p_{n-1}}$ becomes 1 as n tends to infinity.

The formulas for computing $\gamma^U_{n-1,p_{n-1}}$ and γ^U_{n-1,p_n} are

$$\gamma^U_{n-1,p_{n-1}} = 1 - (1 - p_{n-1})(1 - p_{n-1} \cdot \gamma_{n-2,p_{n-1}})^{n-3},$$

$$\begin{aligned}\gamma^U_{n-1,p_n} &= 1 - (1 - p_n)(1 - p_n \cdot \gamma_{n-2,p_n})^{n-3} \\ &\geq 1 - (1 - p_n)(1 - p_n \cdot \gamma_{n-2,p_{n-1}})^{n-3} \quad (\text{since } \gamma_{n-2,p_{n-1}} \geq \gamma_{n-2,p_n}) \\ &\equiv \gamma^C,\end{aligned}$$

respectively. Since the relation

$$\gamma^U_{n-1,p_{n-1}} \geq \gamma^U_{n-1,p_n} \geq \gamma^C$$

holds, we try to estimate the ratio $\gamma^U_{n-1,p_{n-1}}/\gamma^C$ instead of $\gamma^U_{n-1,p_{n-1}}/\gamma^U_{n-1,p_n}$, and use the ratio $(1 - p_{n-1})(1 - p_{n-1} \cdot \gamma_{n-2,p_{n-1}})^{n-3}/\,(1 - p_n)(1 - p_n \cdot \gamma_{n-2,p_{n-1}})^{n-3}$ in order to evaluate $\gamma^U_{n-1,p_{n-1}}/\gamma^C$. Thus, we have

$$\begin{aligned}&(1 - p_{n-1})(1 - p_{n-1} \cdot \gamma_{n-2,p_{n-1}})^{n-3}/(1 - p_n)(1 - p_n \cdot \gamma_{n-2,p_{n-1}})^{n-3} \\ &= \left(1 - \frac{\alpha}{n-2}\right)\left(1 - \frac{\alpha}{n-2}\gamma_{n-2,p_{n-1}}\right)^{n-3} \Big/ \left(1 - \frac{\alpha}{n-1}\right)\left(1 - \frac{\alpha}{n-1}\gamma_{n-2,p_{n-1}}\right)^{n-3} \\ &= \frac{n-1}{n-2}\left(\frac{n-1}{n-2} \cdot \frac{n-2-\alpha\gamma_{n-2,p_{n-1}}}{n-1-\alpha\gamma_{n-2,p_{n-1}}}\right)^{n-3}\end{aligned}$$

and therefore,

$$\lim_{n \to \infty} \frac{\gamma^U_{n-1,p_{n-1}}}{\gamma^C} = 1$$

holds. This implies $\gamma^U_{n-1,p_{n-1}} = \gamma^U_{n-1,p_n}$ and hence $\gamma^U_{n-1,p_{n-1}} = \gamma^U_{n,p_n}$ in the limit state.

Now, by letting

$$\gamma^U_{n-1,p_n}\left(= \gamma^U_{n-1,p_{n-1}}\right) = \gamma^U_{n,p_n} = x$$

for $n \to \infty$ and substituting these into (5.15), we obtain the stationary equation

$$x = \left(1 - \frac{\alpha}{n-2}\right)\left(1 - \frac{\alpha}{n-2}x\right)^{n-2},$$

which converges, as n tends to infinity, to the solution of the same equation as (5.14):

$$1 - x - e^{-\alpha x} = 0.$$

■

Since $p_n = \dfrac{\alpha}{n-1}$ is the probability of an arc to be present between any two vertices, $\alpha = p(n-1)$ can be regarded as the average out-degree of any vertex in G of type $D(n,p)$. Therefore, Theorem 5.18 states that if the out-degree α of a vertex is less than or equal to 1, the upper bound ζ^U_{n,p_n} of the reachability between any two vertices becomes 0, as n grows to infinity. On the other hand, if the out-degree α is greater than 1, ζ_{n,p_n} converges to $\zeta^U_\alpha = x_2$ (> 0).

Now, we present some computational results of Theorem 5.18. First, we show in Figure 5.18 the behavior of $\gamma^U_{n,\frac{\alpha}{n-1}}$ as a function of n for various α. This may indicate that the convergence speed of $\gamma^U_{n,\frac{\alpha}{n-1}}$ is very fast. Figure 5.19 shows the relationship between two upper bounds $\gamma^U_{n,\frac{\alpha}{n-1}}$ and $\zeta^U_{n,\frac{\alpha}{n-1}}$ for four values of α. This confirms us that $\zeta^U_{n,\frac{\alpha}{n-1}}$ is an upper bound of $\gamma^U_{n,\frac{\alpha}{n-1}}$ and is non-increasing, and furthermore, they converge to the same limit.

Figure 5.20 shows ζ^U_α as a function of α. In this figure, the exact reachabilities γ_{n,p_n} in case of $n = 10, 20, 50, 100$ which are computed by the procedure presented in Section 5.3 are also shown for comparison purpose.

From Theorem 5.18, we immediately have the following corollary.

Corollary 5.20: In a random digraph G of type $D\left(n, \dfrac{\alpha}{(n-1)^\beta}\right)$, the limit value ζ^U_β of ζ^U_{n,p_n} when $n \to \infty$ becomes

$$\zeta^U_\beta = \begin{cases} 1 & (\beta < 1) \\ x_2 & (\beta = 1) \\ 0 & (\beta > 1), \end{cases}$$

where x_2 is the solution of equation (5.14), satisfying $0 \le x_2 < 1$. ■

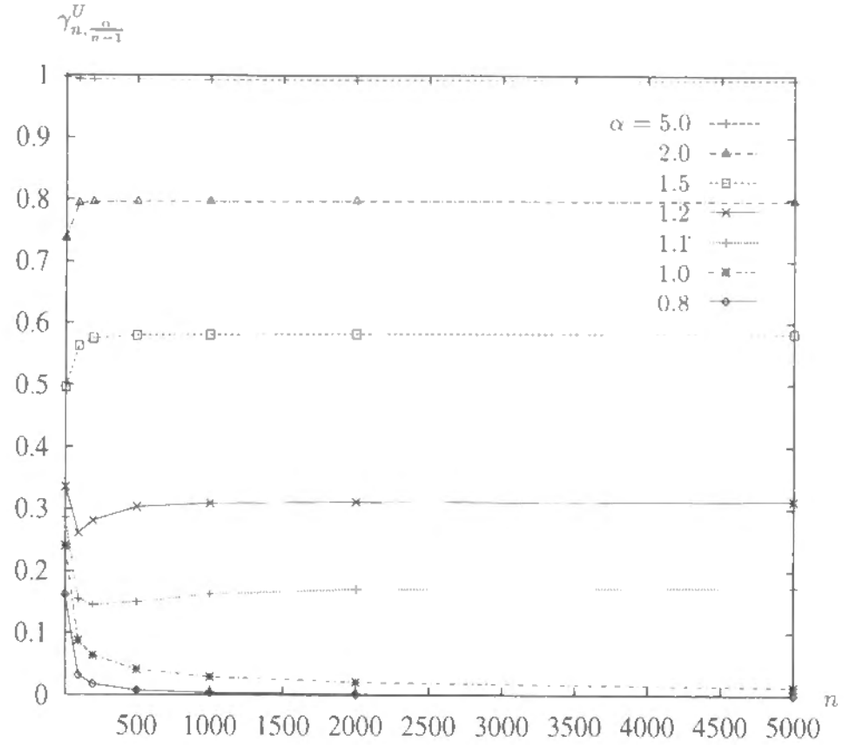
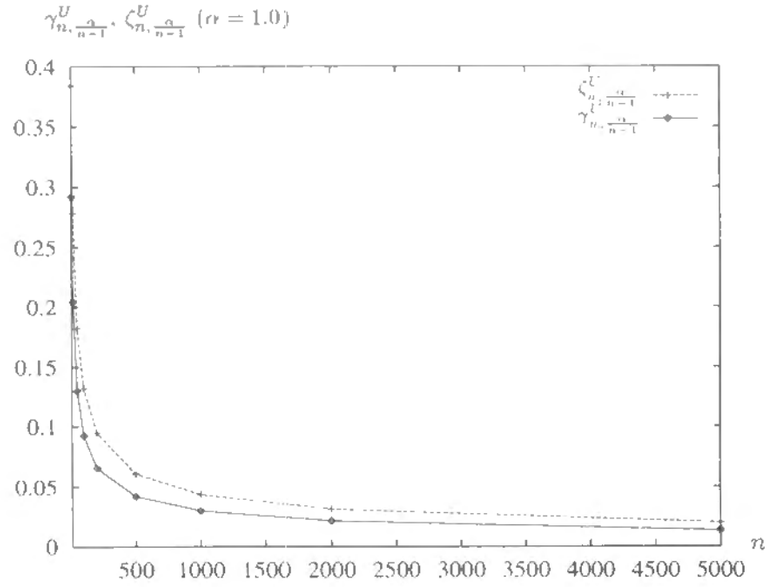
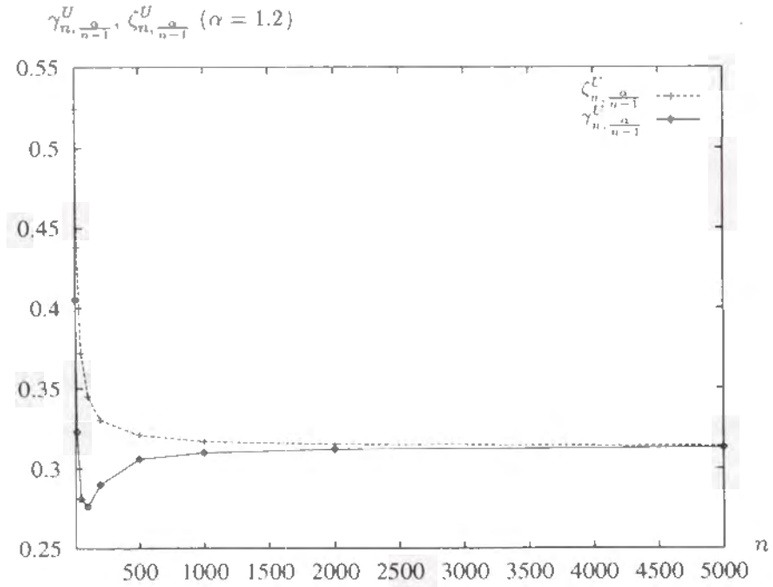


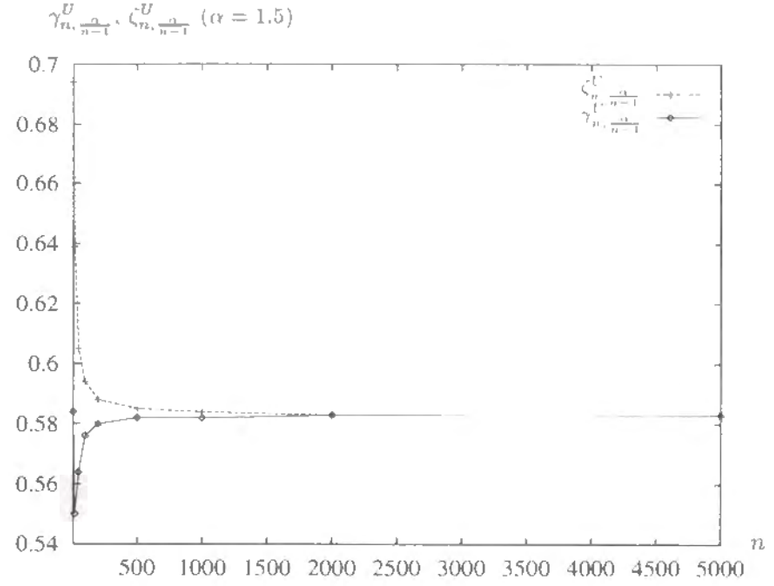
Figure 5.18: Behavior of $\gamma^U_{n,\frac{\alpha}{n-1}}$ as a function of n.

Proof: This result comes from the behavior of the limit value of $\lim_{n\to\infty}\left(1 + \dfrac{1}{(n-1)^\beta}\right)^n$ and Theorem 5.18. ■

This corollary says that if p_n in $D(n,p_n)$ is greater than $O(n^{-1})$, any vertex becomes reachable to all the vertices when n tends to infinity, while if p_n is smaller than $O(n^{-1})$, it becomes reachable to no vertex.







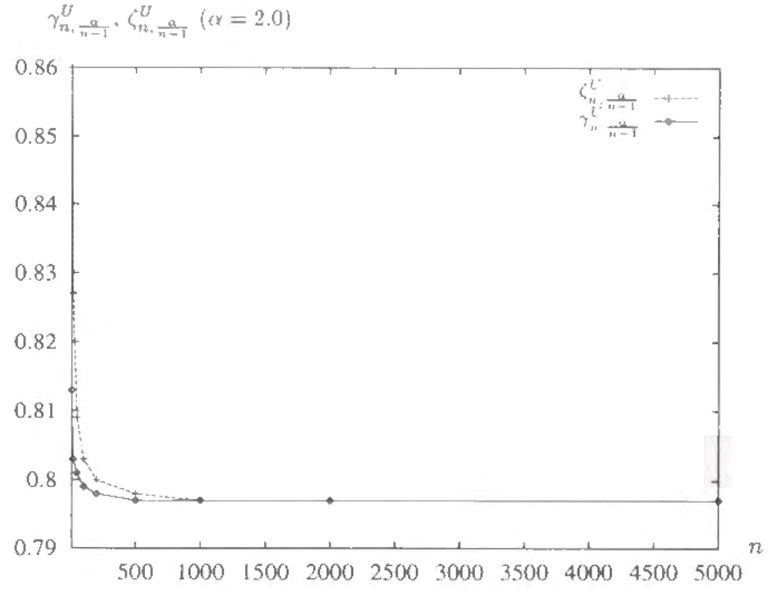


Figure 5.19: Relationship between $\gamma^U_{n,\frac{\alpha}{n-1}}$ and $\zeta^U_{n,\frac{\alpha}{n-1}}$.

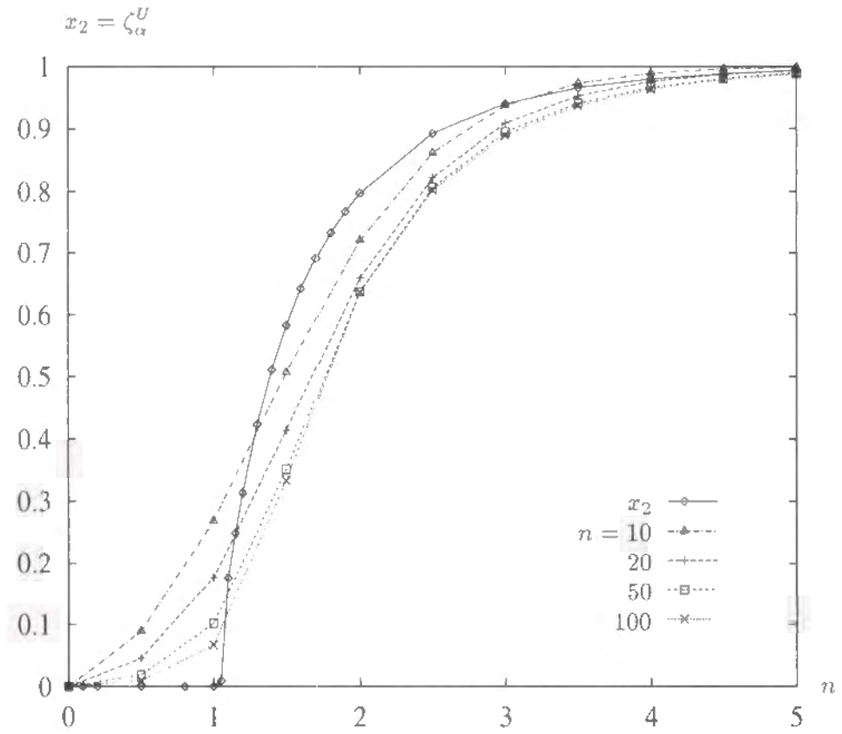


Figure 5.20: The solution x_2 and the exact reachabilities as a function of α.

5.7 On the Number of Reachable Vertices and the Size of Transitive Closure

While $\gamma_{n,p}$ denotes the reachability between any two vertices, $(n-1)\gamma_{n,p}$ denotes the number of reachable vertices from a given vertex, and $n(n-1)\gamma_{n,p} + n\gamma_{n+1,p}$ denotes the expected size of the transitive closure by Theorem 5.7. Therefore, we are also interested in the asymptotic behavior of those values or their upper bounds, when n tends to infinity.

5.7.1 Asymptotic Behavior of the Number of Reachable Vertices

According to the definition of the reachability γ_{n,p_n}, $(n-1)\gamma_{n,p_n}$ represents the expected number of reachable vertices from one vertex in a random graph of type $D(n,p_n)$. In this subsection, we consider its appropriate upper bound and analyze the behavior in case n tends to infinity. Then, we have the following theorem.

Theorem 5.21: The $(n-1)\zeta^U_{n,p_n}$ gives an upper bound on the expected number of reachable vertices from one vertex in a random digraph of type $D(n,p_n)$, where the upper bound ζ^U_{n,p_n} on the reachability is computed by (5.8), that is,

$$\zeta^U_{n,p_{n-1}} = 1 - (1-p_{n-1})(1 - p_{n-1}\cdot\zeta^U_{n-1,p_{n-1}})^{n-2}. \tag{5.16}$$

Furthermore, if $p_n = \dfrac{\alpha}{n-1}$, $(n-1)\zeta^U_{n,p_n}$ converges as follows:

$$\lim_{n\to\infty}(n-1)\zeta^U_{n,\frac{\alpha}{n-1}} = \frac{\alpha}{1-\alpha},$$

where α satisfies $0 \le \alpha < 1$. ■

Proof: First of all, it is obvious that $(n-1)\zeta^U_{n,p_n}$ is an upper bound on the expected number of the reachable vertices from one vertex, because $(n-1)\gamma_{n,p_n}$ expresses the exact expected number of the reachable vertices from a given vertex, and ζ^U_{n,p_n} is an upper bound on γ_{n,p_n}, that is,

$$(n-1)\gamma_{n,p_n} \le (n-1)\gamma^U_{n,p_n} \le (n-1)\zeta^U_{n,p_n}$$

holds. Now, define

$$y_n \equiv (n-1)\zeta^U_{n,p_n}.$$

Then, by substituting $\zeta_{n,p_n} = \dfrac{y_n}{n-1}$ and $p_n = \dfrac{\alpha}{n-1}$ into formula (5.16), we have

$$\begin{aligned}\frac{y_n}{n-1} &= 1 - \left(1 - \frac{\alpha}{n-1}\right)\left(1 - \frac{\alpha}{n-1}\cdot\frac{y_{n-1}}{n-1}\right)^{n-2} \\ &= 1 - \left(1 - \frac{\alpha}{n-1}\right)\left(1 - \alpha\frac{n-2}{(n-1)^2}y_{n-1} + \frac{\alpha^2}{2}\frac{(n-2)(n-3)}{(n-1)^4}y^2_{n-1} - \cdots\right),\end{aligned}$$

that is,

$$\begin{aligned}y_n &= (n-1) - (n-1)\left(1 - \alpha\frac{1}{n-1} - \alpha\frac{(n-2)}{(n-1)^2}y_{n-1} + \alpha^2\frac{(n-2)}{(n-1)^3}y_{n-1}\right. \\ &\qquad \left. + \frac{\alpha^2}{2}\frac{(n-2)(n-3)}{(n-1)^4}y^2_{n-1} - \frac{\alpha^3}{2}\frac{(n-2)(n-3)}{(n-1)^5}y^2_{n-1} - \cdots\right) \\ &= \alpha + \alpha\frac{n-2}{n-1}y_{n-1} - \alpha^2\frac{n-2}{(n-1)^2}y_{n-1} \\ &\qquad - \frac{\alpha^2}{2}\frac{(n-2)(n-3)}{(n-1)^3}y^2_{n-1} + \frac{\alpha^3}{2}\frac{(n-2)(n-3)}{(n-1)^4}y^2_{n-1} - \cdots.\end{aligned}$$

In this formula, since all the terms after the second one become negligible when n tends to infinity, it becomes

$$y_n = \alpha + \alpha\cdot\frac{n-2}{n-1}\cdot y_{n-1} + O\left(\frac{y_{n-1}}{n}\right). \tag{5.17}$$

Therefore, since

$$\lim_{n\to\infty}\left(\alpha\cdot\frac{n-2}{n-1}\cdot y_{n-1}\right) = \alpha y_{n-1},$$

formula (5.17) becomes

$$y_n = \alpha + \alpha y_{n-1}$$

in the limit state of $n \to \infty$. From this, we have

$$y_n - \frac{\alpha}{1-\alpha} = \alpha\left(y_{n-1} - \frac{\alpha}{1-\alpha}\right), \tag{5.18}$$

and this implies that the sequence $\left\{y_n - \dfrac{\alpha}{1-\alpha}\right\}$ converges to 0 if α satisfies $0 \le \alpha < 1$. In other words, for $0 \le \alpha < 1$,

$$y = \lim_{n\to\infty} y_n = \lim_{n\to\infty}(n-1)\zeta^U_{n,\frac{\alpha}{n-1}} = \frac{\alpha}{1-\alpha}.$$

(Here, the recursive formula (5.18) is also satisfied by the condition that

$$y_2 = y_3 = \cdots = \frac{\alpha}{1-\alpha}.$$

But this cannot be realized in our case because, strictly speaking, y_n includes higher terms of $O\left(\dfrac{y_{n-1}}{n}\right)$ in formula (5.17).) This concludes the proof of this theorem. ■

Now, we show some numerical results for the expected number of reachable vertices $(n-1)\gamma_{n,\frac{\alpha}{n-1}}$ and its upper bound $(n-1)\zeta^U_{n,\frac{\alpha}{n-1}}$. The first figure in Figure 5.21 shows the limit value $\dfrac{\alpha}{1-\alpha}$ of $(n-1)\zeta^U_{n,\frac{\alpha}{n-1}}$ and the exact number of reachable vertices $(n-1)\gamma_{n,\frac{\alpha}{n-1}}$ for some finite n's as a function of α $(0 \le \alpha \le 0.9)$, and the second figure in Figure 5.21 is its expansion in the range of $0 \le \alpha \le 0.5$. In Figure 5.22, we illustrate the relationship among $(n-1)\gamma_{n,\frac{\alpha}{n-1}}$, $(n-1)\zeta^U_{n,\frac{\alpha}{n-1}}$ and the limit value $\dfrac{\alpha}{1-\alpha}$ as functions of n, for $\alpha = 0.2, 0.5$ and 0.8.

All of these figures verify that $(n-1)\zeta^U_{n,\frac{\alpha}{n-1}}$ is an upper bound on $(n-1)\gamma_{n,\frac{\alpha}{n-1}}$, and is converging to its limit value $\dfrac{\alpha}{1-\alpha}$.

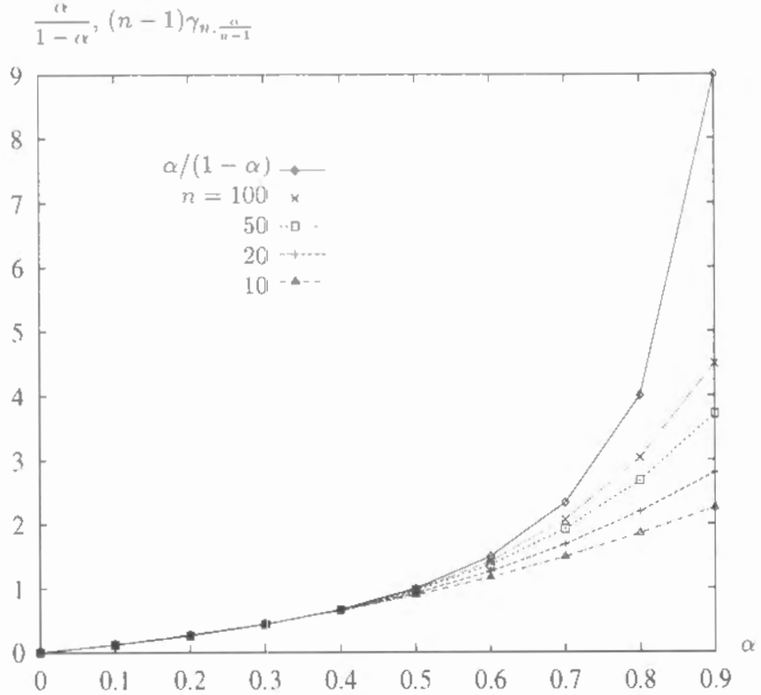
Now, we can present here a trivial property as a corollary, which is derived directly from the proof of Theorem 5.21.

Corollary 5.22: In a random digraph G of type $D\left(n, \dfrac{\alpha}{(n-1)^\beta}\right)$, the limit value y^U_β of $(n-1)\zeta^U_{n,\frac{\alpha}{(n-1)^\beta}}$ when $n \to \infty$ becomes

$$y^U_\beta = \begin{cases} \infty & (\beta < 1) \\ \infty & (\beta = 1, \alpha > 1) \\ \dfrac{\alpha}{1-\alpha} & (\beta = 1, 0 \le \alpha < 1) \\ 0 & (\beta > 1), \end{cases}$$

according to the values of α and β. ■

Proof: This is mainly because the sequence $\left\{y_n - \dfrac{\alpha}{1-\alpha}\right\}$ in formula (5.18) diverges as n tends to infinity when $\alpha > 1$. ■



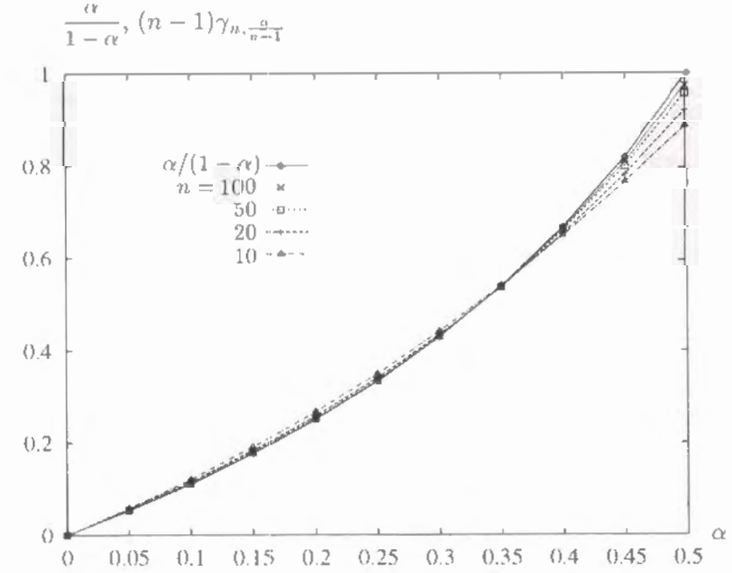
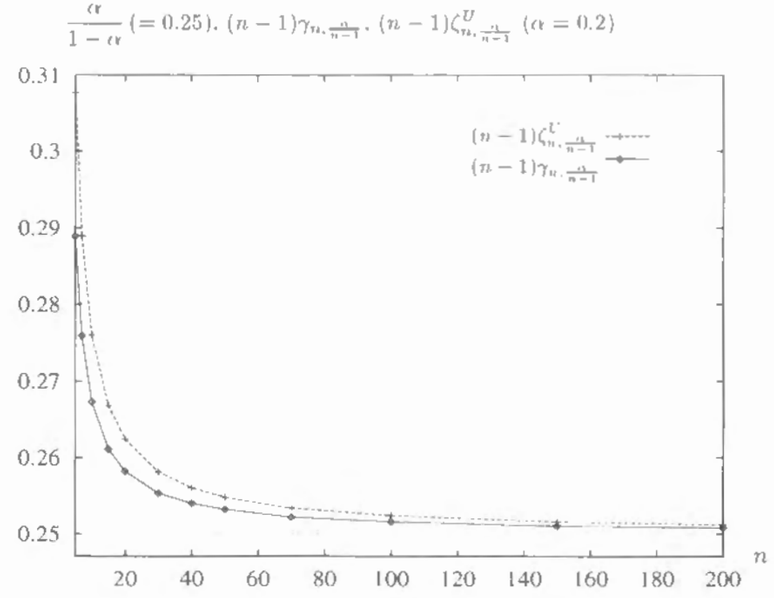
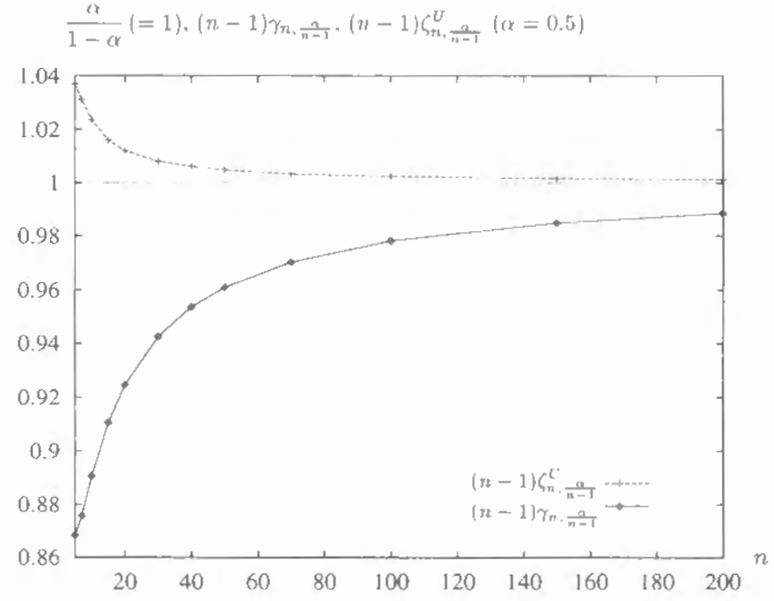


Figure 5.21: The limit value y and the exact expected number of reachable vertices.





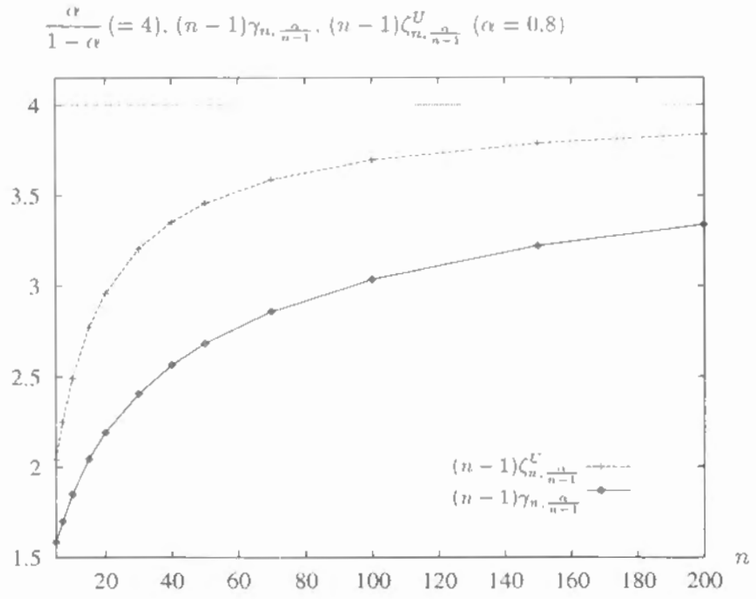


Figure 5.22: Relationship among $(n-1)\gamma_{n,\frac{\alpha}{n-1}}$, $(n-1)\zeta^U_{n,\frac{\alpha}{n-1}}$ and the limit value $\frac{\alpha}{1-\alpha}$.

5.7.2 Asymptotic Behavior of the Size of Transitive Closure

Along a similar line as in the discussion in the previous subsection, we discuss in this subsection the asymptotic behavior of the size of the transitive closure of a random digraph.

By Lemma 5.7, we follow the rule to denote the expected size of the transitive closure of a random digraph of type $D_L(n,p_n)$ by $|g^+_L(n,p_n)|$. It is computed by

$$|g^+_L(n,p_n)| = n(n-1)\gamma_{n,p_n} + \gamma_{n+1,p_n}, \tag{5.19}$$

when p is given by p_n as a function of n.

For the purpose of estimating an upper bound on $|g^+_L(n,p_n)|$, we extend the definition of ζ^U_{n,p_n} to ζ^U_{n+n',p_n}, in which $n+n'$ is greater than n ($n' \geq 1$), that is,

$$\begin{cases} \zeta^U_{n_0+n',p_{n_0}} = \gamma^U_{n_0+n',p_{n_0}}, \\ \zeta^U_{n+n',p_n} = 1-(1-p_{n-1})(1-p_{n-1}\cdot\zeta^U_{n+n'-1,p_{n-1}})^{n-2} \quad (n > n_0). \end{cases} \tag{5.20}$$

It is obvious that ζ^U_{n+n',p_n} is an upper bound on γ^U_{n+n',p_n}, and therefore, on the reachability γ_{n+n',p_n}.

According to this definition and formula (5.19), we have the following theorem.

Theorem 5.23: The $n^2\zeta^U_{n+1,p_n}$ is an upper bound on the expected size of the transitive closure of a random digraph of type $D_L(n,p_n)$, and when p_n is given as $p_n = \frac{\alpha}{n^2}$, $n^2\zeta^U_{n+1,\frac{\alpha}{n^2}}$ converges to α, that is,

$$\lim_{n\to\infty} n^2\zeta^U_{n,\frac{\alpha}{n^2}} = \alpha,$$

where α satisfies $\alpha \geq 0$. ■

Proof: Since the expected size $|g^+_L(n,p_n)|$ of the transitive closure of a random digraph of type $D_L(n,p_n)$ is

$$|g^+_L(n,p_n)| = n(n-1)\gamma_{n,p_n} + n\gamma_{n+1,p_n}$$

by Lemma 5.7, it is bounded from above as

$$\begin{aligned} |g^+_L(n,p_n)| &\leq n(n-1)\gamma_{n+1,p_n} + n\gamma_{n+1,p_n} \quad (\text{since } \gamma_{n+1,p_n} \geq \gamma_{n,p_n}) \\ &= n^2\gamma_{n+1,p_n} \\ &\leq n^2\gamma^U_{n+1,p_n} \\ &\leq n^2\zeta^U_{n+1,p_n}. \end{aligned}$$

Therefore, $n^2\zeta^U_{n+1,p_n}$ is an upper bound on $|g^+_L(n,p_n)|$. Now, we examine the asymptotic behavior of $n^2\zeta^U_{n+1,p_n}$.

Define

$$z_{n+1} \equiv n^2\zeta^U_{n+1,p_n}.$$

Then, by substituting $\zeta^U_{n+1,p_n} = \frac{z_{n+1}}{n^2}$ and $p_n = \frac{\alpha}{n^2}$ into formula (5.20), we have

$$\begin{aligned}\frac{z_{n+1}}{n^2} &= 1 - \left(1 - \frac{\alpha}{n^2}\right)\left(1 - \frac{\alpha}{n^2} \cdot \frac{z_n}{(n-1)^2}\right)^{n-2} \\ &= 1 - \left(1 - \frac{\alpha}{n^2}\right)\left(1 - \alpha\frac{n-2}{n^2(n-1)^2}z_n + \frac{\alpha^2}{2}\frac{(n-2)(n-3)}{n^4(n-1)^4}z_n^2 - \cdots\right),\end{aligned}$$

that is,

$$\begin{aligned}z_{n+1} &= n^2 - (n^2 - \alpha)\left(1 - \alpha\frac{n-2}{n^2(n-1)^2}z_n + \frac{\alpha^2}{2}\frac{(n-2)(n-3)}{n^4(n-1)^4}z_n^2 - \cdots\right) \\ &= n^2 - \left(n^2 - \alpha - \alpha\frac{n^2(n-2)}{n^2(n-1)^2}z_n + \alpha^2\frac{(n-2)}{n^2(n-1)^2}z_n\right. \\ &\qquad \left. + \frac{\alpha^2}{2}\frac{n^2(n-2)(n-3)}{n^4(n-1)^4}z_n^2 - \frac{\alpha^3}{2}\frac{n^2(n-2)(n-3)}{n^4(n-1)^4}z_n^2 - \cdots\right) \\ &= \alpha + \alpha\frac{(n-2)}{(n-1)^2}z_n - \alpha^2\frac{(n-2)}{n^2(n-1)^2}z_n \\ &\qquad - \frac{\alpha^2}{2}\frac{(n-2)(n-3)}{n^2(n-1)^4}z_n^2 + \frac{\alpha^3}{2}\frac{(n-2)(n-3)}{n^4(n-1)^4}z_n^2 - \cdots.\end{aligned}$$

In this formula, since all the terms except the first one become 0 as n tends to infinity, we obtain

$$z = \lim_{n\to\infty} z_{n+1} = \lim_{n\to\infty} n^2\zeta_{n+1,\frac{\alpha}{n^2}} = \alpha$$

for the initial probability $p_n = \frac{\alpha}{n^2}$ (for $\alpha \geq 0$). ■

Here, we present some numerical results. Figure 5.23 shows the exact size $n(n-1)\gamma_{n,\frac{\alpha}{n^2}} + n\gamma_{n+1,\frac{\alpha}{n^2}}$ of transitive closure and the limit value α for various n and α. We illustrate in Figure 5.24 the relationship among $n(n-1)\gamma_{n,\frac{\alpha}{n^2}} + n\gamma_{n+1,\frac{\alpha}{n^2}}$, $n^2\zeta^U_{n+1,\frac{\alpha}{n^2}}$ and α for $\alpha = 0.5$ and 2.0 as a function of n. These justify the results of Theorem 5.23.

Finally, we have a corollary similar to Corollary 5.22.

Corollary 5.24: In a random digraph G of type $D\left(n, \frac{\alpha}{n^\beta}\right)$, the limit value z^U_β of $n^2\zeta^U_{n+1,\frac{\alpha}{n^\beta}}$ when $n \to \infty$ becomes

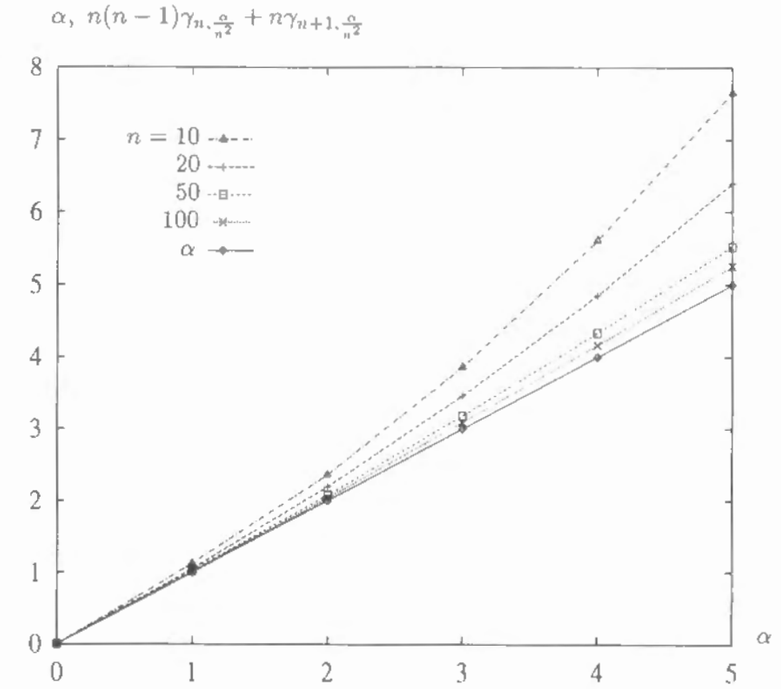


Figure 5.23: The exact size $n(n-1)\gamma_{n,\frac{\alpha}{n^2}} + n\gamma_{n+1,\frac{\alpha}{n^2}}$ of the transitive closure and its limit value α.

$$z^U_\beta = \begin{cases} \infty & (\beta < 2) \\ \alpha & (\beta = 2) \\ 0 & (\beta > 2), \end{cases}$$

according to β. ■

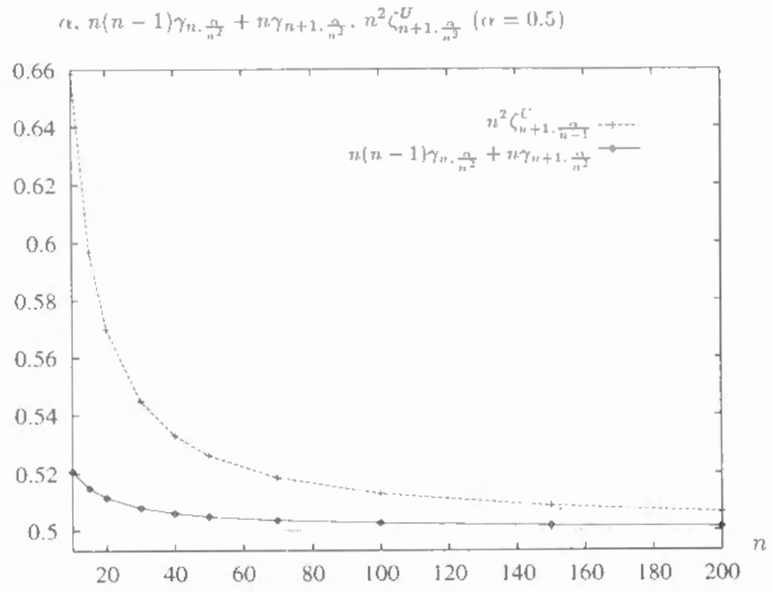


Figure 5.24: Relationship among $n(n-1)\gamma_{n,\frac{\alpha}{n^2}} + n\gamma_{n+1,\frac{\alpha}{n^2}}$, $n^2\zeta^U_{n+1,\frac{\alpha}{n^2}}$ and the limit value α.

5.8 Conclusion

In this chapter, we illustrated that the expected size of the transitive closure of a random graph of type $D(n,p)$ can be computed from the reachability $\gamma_{n,p}$. Then, we proposed two different methods of computing $\gamma_{n,p}$ exactly. One is to count the number of reachable vertices by the length of their shortest paths from a source vertex, and the other is to utilize the conditional reachabilities.

Since it takes much time to compute exact reachability $\gamma_{n,p}$, we presented a convenient and fast way to obtain lower and upper bounds on the reachability, based on the second approach. Although the lower bound is rather rough, the upper bound is fairly close to the exact reachability. These bounds are useful to estimate lower and upper bounds of the cost for computing transitive closures of relations, as described in Chapters 3 and 4.

Then, we examined the asymptotic behavior of the upper bound ζ^U_{n,p_n} of the reachability γ_{n,p_n}. The result shows that the limit value ζ^U_α of ζ^U_{n,p_n} depends on the out-degree α of a vertex. If α is a constant, it approaches to the largest solution x_2 of the equation $1 - x - e^{-\alpha x} = 0$, and if α is a function of n, it approaches 0 or 1.

Furthermore, while γ_{n,p_n} denotes the reachability between any two vertices, $(n-1)\gamma_{n,p_n}$ denotes the number of reachable vertices from a given vertex, and $n(n-1)\gamma_{n,p_n} + n\gamma_{n+1,p_n}$ denotes the number of all the pairs of reachable vertices, i.e., the size of the transitive closure of a given random digraph. We also examined the behaviors of these values. Although we could investigate the asymptotic behaviors of upper bounds $(n-1)\zeta^U_{n,p_n}$ and $n^2\zeta^U_{n+1,p_n}$ only, they seem to converge respectively to the exact values $(n-1)\gamma^U_{n,p_n}$ and $n(n-1)\gamma_{n,p_n} + n\gamma_{n+1,p_n}$ in the limit state.

Chapter 6

Complexity of the Optimum Join Order Problem

6.1 Introduction

Relational databases are not only used as the systems in the real world, but also used as subsystems of more complex systems, e.g., deductive database systems [Ban 86a, Bay 85b, Gard 89, Kif 86, Miy 89]. Optimization of processing database queries is also important in order to increase their applicability.

As we viewed in Chapter 2, the process of handling database queries can be decomposed into basic operations of relational algebra [Gard 89, Ull 89b]. Among the basic operations in relational algebra, joins [Gra 93, Ibara 84, Kim 80, Kri 86] are the most important in the sense that they consume most of the computational time. Therefore, a considerable number of recent studies have been made to minimize the computational cost associated with joins. However, most of them are based on empirical argument, and there is little theoretical work done in this area.

Among the existing algorithms for computing many joins, the nested-iteration algorithm [Kim 80] is straightforward but it is not efficient. More efficient ones are the

nested-loops algorithm [Gra 93, Ibara 84, Kim 80, Kri 86] and the merge-scan algorithm [Gra 93, Kri 86] which are well-known. As the nested-loops algorithm, a polynomial time algorithm is known [Ibara 84] that finds an optimal nesting order of u joins in a given query tree, which minimizes the cost of joins within the framework of nested-loops algorithm. For these works, some papers discussed the optimum order of joins executed by the nested-loops algorithm [Ibara 84], however, the merge-scan algorithm has so far received limited attention [Kri 86] in the theoretical sense in spite of its efficiency in computing joins. (The results for the two algorithms are not contradictory because they are based on different methods of computing joins, and the optimization is attempted only within the framework of each algorithm. The optimum costs with these two algorithms of computing joins may be different.)

Therefore, in this chapter, we concentrate on the merge-scan algorithm and try to minimize the total size of intermediate relations generated in processing the whole joins. The total size is important and it is often adopted as a typical measure because it represents the memory space requirement of the entire computation. Moreover, it may be viewed as an approximate measure of the time complexity.

For the purpose of formalizing our problem, we use the query graph (tree), which only focuses on the attributes and the selectivities of relations on which joins are executed. It will be shown in this chapter that determining a join order of the merge-scan algorithm that minimizes the above cost is equivalent to finding an optimum selection order of all edges in a query graph. Then we examine the computational complexity of the problem of finding an optimum selection order, and show that it is NP-hard [Gar 79] even if query graphs are restricted to be trees. This fact suggests that no algorithm can compute the optimum join order in polynomial time. However, it is also found out that, if we further restrict query trees to certain special types, relatively efficient polynomial time algorithms exist.

6.2 Optimum Join Order Problem

6.2.1 Join Algorithms

We illustrate two typical algorithms for computing joins in this subsection, whose outlines are introduced in Chapter 2. That is, the nested-loops algorithm and the merge-scan algorithm, for computing a single join $R_1 \underset{A_p=B_q}{\bowtie} R_2$, where R_1 and R_2 consist of r_1 tuples $t_{11}, \ldots, t_{1r_1}$ and r_2 tuples $t_{21}, \ldots, t_{2r_2}$, respectively.

Nested-Loop Algorithm

The nested-loop algorithm to compute $R = R_1 \underset{A_p=B_q}{\bowtie} R_2$ is given in Program 6.1. Similar

```
Procedure Nested-Loop1
Input: R1, R2
Output: R = R1 ⋈(Ap=Bq) R2
1   begin
2   R := φ;
3   for i := 1 to r1 do
4     begin
5     for j := 1 to r2 do
6       begin
7       if t1i.Ap = t2j.Bq then
8         R := R ∪ {(ti, tj)};
9       end
10    end
11  end.
```

Program 6.1: The nested-loop algorithm.

to the definition of natural join between two relations, (t_i, t_j) denotes the resulting tuple of the natural join of tuples t_i and t_j instead of relations R_1 and R_2 between the specified attributes A_p and B_q for R_1 and R_2, namely, $R = \{(t_i, t_j) \mid t_i.A_p = t_j.B_q\}$. This algorithm

finds all the tuples in R_2 whose values of component of attribute B_q coincides with the value of each component of attribute A_i in R_1.

This algorithm can be generalized to compute the joins among n relations. For this purpose, it is nested as shown in Program 6.2. The "nested-loop" is named for this reason. In this algorithm, we assume for simplicity that the join attribute in R_k has appeared

```
Procedure Nested-Loop2
Input: R_1, ..., R_n
Output: R = (((R_1 ⋈ R_2) ⋈ ···) ⋈ R_{n-1}) ⋈ R_n
1     begin
2     R := φ;
3     for i_1 := 1 to r_1 do
4       begin
5       for i_2 := 1 to r_2 do
6         if (the value of join attribute in R_2 matches
            the value of join attribute in R_1) then
7         begin
            ⋮
8           for i_n := 1 to r_n do
9             if (the value of join attribute in R_n matches the value
                of join attribute either in R_1, ..., R_{n-1}) then
10            begin
11              R := R ∪ {(t_{1i_1}, ..., t_{ni_n})};
12            end
            ⋮
13        end
14      end
15    end.
```

Program 6.2: The modified nested-loop algorithm.

once either in the previous relations $R_1, \ldots, R_{k-1}$. Similar to the procedure Nested-Loop1, $r_1, \ldots, r_n$ is the sizes of relations $R_1, \ldots, R_n$, and $(t_{1i_1}, \ldots, t_{ni_n})$ denotes the

resulting tuple of the natural join of tuples $t_{1i_1}, \ldots, t_{ni_n}$ by the specified join attributes.

Merge-Scan Algorithm

The merge-scan algorithm for computing a single join is described in Program 6.3. This

```
Procedure Merge-Scan1
Input: R_1, R_2
Output: R = R_1 ⋈_{A_p=B_q} R_2
1     begin
2     R := φ;
3     Sort all the tuples in R_1 on A_p in the increasing order;
4     Sort all the tuples in R_2 on A_q in the increasing order;
5     i := 1; j := 1;
6     while (i ≤ r_1 and j ≤ r_2) do
7       begin
8       if t_i.A_p > t_j.A_q then j := j + 1;
9       else if t_i.A_p < t_j.A_q then i := i + 1;
10      else do (t_i.A_p = t_j.B_q)
11        begin
12          k := 0;
13          while (j + k ≤ r_2 and t_j.A_p = t_{j+k}.A_q) do
14            begin
15              R := R ∪ {(t_i, t_j)};
16              k := k + 1;
17            end
18          i := i + 1;
19        end
20    end.
21  end.
```

Program 6.3: The merge-scan algorithm.

algorithm firstly sorts all the tuples in R_1 and R_2 by the join attributes A_p and B_q, respectively, in the increasing order, and scans those sorted tuples in R_1 and R_2 simul-

taneously in this order of values of components in B_q for each value of component in A_p. Then, it merges the tuple t_i in R_1 and t_j in R_2, whose values in the join attributes coincide, into a new tuple (t_i, t_j).

This algorithm can easily applied to compute the joins among n relations. Suppose there are m joins to compute among n relations. In other words, there are m pairs of two relations which have common join attributes each other. This algorithm can begin with computing one of these m pairs, and all the m joins one after another, in which all the m joins are computed by the merge-scan algorithm between two relations. The detail will be described in the subsequent subsections.

6.2.2 Complexity of Computing Joins with Selectivities

As we noticed in Chapter 2, we estimate the complexity of computing joins by the representative two algorithms, the nested-loop algorithm and the merge-scan algorithm, and we define the computing costs of them. When considering the complexity of computing joins, there may be two measures to estimate it as we saw in Section 5 in Chapter 2. (That is, (1) the number of tuples accessed during the computation of the join, or (2) the number of tuples outputted explicitly as the result of the join.)

By using these measures, we try to estimate the complexity of computing the join $R_1 \underset{A_p=B_q}{\bowtie} R_2$ with their selectivity s_{12} by the nested-loop and the merge-scan algorithms.

Nested-Loop Algorithm

1. Since it accesses r_2 tuples in inner loop for each of r_1 tuples in outer loop, it requires $O(r_1 r_2)$ computational time for accessing tuples.
2. Since the resulting size of the join is $s_{12} r_1 r_2$, it requires $O(s_{12} r_1 r_2)$ computational time for outputting tuples.

Merge-Scan Algorithm

1. The computational complexity is $O(r_1 \log r_1 + r_2 \log r_2)$ to sort R_1 and R_2, and $O(r_1 + r_2)$ to merge R_1 and R_2. The total complexity is $O(r_1 \log r_1 + r_2 \log r_2 + r_1 + r_2)$ to access tuples.
2. It requires $O(s_{12} r_1 r_2)$ computational time for outputting tuples, similar to the nested-loop procedure.

Since it is usually regarded that the unit time required for outputting the resulting relation explicitly costs more than that for accessing tuples, we decided to adopt the complexity $O(s_{12} r_1 r_2)$ as the cost of computing the join $R_1 \underset{A_p=B_q}{\bowtie} R_2$. Moreover, since we consider only merge-scan algorithm for computing joins in this chapter, the term $O(s_{12} r_1 r_2)$ is not only the size of the resulting relation but becomes dominant in the entire computation if r_1 and r_2 are large.

Consequently, we define that the cost $c(R_1, R_2)$ of joining R_1 and R_2 of sizes r_1 and r_2 and the selectivity s_{12} between R_1 and R_2 as

$$c(R_1, R_2) \equiv s_{12} r_1 r_2.$$

As discussed above, this is the size of the resulting relation, which is invariant whatever kind of algorithms is used for computing the join.

6.2.3 Joins among Many Relations by the Merge-Scan Algorithm

The problem here is to compute all joins of u pairs of relations chosen from a set of v relations R_1, R_2, ... ,R_v. There may be relations used more than once, which are executed over different or identical attributes. But we avoid to show the join attributes between each pair of two relations, for convenience.

For the use of this purpose, we can modify the procedure Merge-Scan1 into the procedure Merge-Scan2, and it is shown in Program 6.4. In this procedure, the notation R_{*i_1*} implies the resulting relation, in which the relation R_{i_1} is joined, or a single relation R_{i_1} itself.

```
Procedure Merge-Scan2
Input: R_1, ..., R_v, Joins of u pairs E = {e_1, ..., e_u} of relations
Output: The resulting relation of computing all u joins
1   begin
2   while E ≠ φ do
3     begin
4       Choose any pair e_i = (R_{i_1}.A_{i_p}, R_{i_2}.B_{i_q}) in E;
5       Sort the relation R_{w_{i_1}*};
6       Sort the relation R_{w_{i_2}*};
7       Compute R_{i_1} ⋈_{A_{i_p}=B_{i_q}} R_{i_2};
8       E = E\e_i;
9     end
10  end.
```

Program 6.4: The modified merge-scan algorithm.

The merge-scan algorithm can begin at any pair of relations (any join) among u pairs, and output the result as an explicit temporary relation. The subsequent joins are not necessarily executed on the pair of relations one of which is the temporary relation just computed in the previous selection of a pair. In other words, the merge-scan algorithm may produce one or more temporary relations.

Suppose that the merge-scan algorithm firstly selects one pair of relations to join, say R_i and R_j, and constructs the resulting relation R_{ij} of size $s_{ij}r_ir_j$. As a result, we are given a list of $(u-1)$ joins. The original joins between some R_k and either R_i or R_j are then considered as joins between R_k and R_{ij}. Then, a pair of relations among $(u-1)$ pairs is again selected to execute a join. This procedure is repeated until all joins are computed. As we shall see later, the order of joins executed in the above process is crucial in determining the sum of intermediate relation sizes.

It is noted here that selectivities are assumed to be invariant during the above process

of merging relations by joins. For example, selectivity s_{ik} between R_i and R_k is assumed to give the selectivity between R_{ij} and R_k after joining R_i and R_j. In general, this invariance does not hold, since the probability distribution of values in an attribute A_p changes if a join is executed on A_p. However, if we assume the equal probability of all values, as was assumed to obtain formula (2.7), it is easily shown that the resulting attribute also has the equal probability for all values, and the invariance of selectivity is maintained. Another special case is when no two joins use the same attribute. In this case, if the values in each tuple of a table are generated independently, the invariance is preserved even for a more general case of formula (2.8). These special cases of practical importance may justify our assumption on the selectivity invariance.

6.2.4 Query Graph Representation of the Merge-Scan Algorithm

When we are asked to compute a list of u joins on v relations R_1, R_2, ..., R_v, we can make a query graph $G = (V, E)$, whose set of vertices V consists of v relations R_1, ..., R_v and whose set of edges E consists of u pairs of joins between two relations R_i and R_j, i.e., $e = (R_i, R_j)$.

In this case, instead of giving join attributes between relations R_i and R_j, we give the selectivity s_{ij} on each edge. We can regard that a query tree $T = (V, E)$ is the special case of a query graph which consists of v vertices and $u = v - 1$ edges.

Applying a join between R_i and R_j can be regarded as selecting the corresponding edge $e = (R_i, R_j)$ and merging its end nodes into R_{ij}, whose size then becomes

$$c(e) = s_{ij}r_ir_j. \tag{6.1}$$

The edge $e = (R_i, R_j)$ is removed from the graph, but other edges, including those adjacent to R_i or R_j, remain in the graph with the same selectivities. Therefore, determining the order of u joins is visualized as selecting successively u edges in G. Upon merging all edges, G becomes a graph consisting only of a single node.

Example 6.1: Consider the situation that we have to answer the query

$$\texttt{query}(\texttt{X}_4,\texttt{X}_5,\texttt{X}_7) \leftarrow \texttt{r}_1(\texttt{X}_1,\texttt{X}_2,\texttt{X}_3),\texttt{r}_2(\texttt{X}_1,\texttt{X}_4),\texttt{r}_3(\texttt{X}_2,\texttt{X}_5),\texttt{r}_4(\texttt{X}_3,\texttt{X}_6),\texttt{r}_5(\texttt{X}_6,\texttt{X}_7),$$

where r_i's are EDBs and let $r_1 = 300$, $r_2 = 200$, $r_3 = 100$, $r_4 = 500$ and $r_5 = 400$. The resulting query tree with appropriate selectivities is shown in Figure 6.1. We have

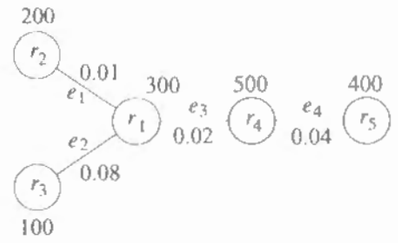


Figure 6.1: An example query tree with selectivities.

$4! = 24$ possible permutations of the orders to execute joins, because there are four edges (joins), $e_1 = (r_1, r_2)$, $e_2 = (r_1, r_3)$, $e_3 = (r_1, r_4)$ and $e_4 = (r_4, r_5)$ in the query tree. If we happen to choose a permutation

$$\pi = (e_2, e_4, e_1, e_3),$$

the computing cost c of the total join is derived as follows:

$$c_1 = c(e_2) = 0.08 \times 300 \times 100 = 2400,$$

$$c_2 = c(e_4) = 0.04 \times 500 \times 400 = 8000.$$

Since node r_3 is merged into r_1, and r_5 into r_4 by these joins, we then have

$$c_3 = c(e_1) = 0.01 \times 2400 \times 200 = 4800,$$

$$c_4 = c(e_3) = 0.02 \times 4800 \times 8000 = 384000.$$

Therefore, the total cost c is

$$c = c_1 + c_2 + c_3 + c_4 = 399200.$$

■

6.2.5 Problem OPTJOIN

The above discussion leads to the problem of finding an optimal join order (i.e., finding an optimal order of selecting edges) for a given query tree. We call this problem OPTJOIN, and it is formalized as followings.

OPTJOIN

Input: a query tree $T = (V, E)$, $V = \{R_1, R_2, \ldots, R_v\}$, $E = \{e_1, e_2, \ldots, e_u\}$ $(u = v - 1)$, the size r_i of each node R_i, and the selectivity s_{ij} of each edge $e = (R_i, R_j)$.

Output: a permutation $\pi = (e_{i_1}, e_{i_2}, \ldots, e_{i_u})$ of all edges in E that mininizes the size $C(\pi)$, where

$$C(\pi) = \sum_k c_\pi(e_{i_k}) \tag{6.2}$$

and $c_\pi(e_{i_k})$ is the size (6.1) of joining two end nodes of e_{i_k} after having merged edges $e_{i_1}, e_{i_2}, \ldots, e_{i_{k-1}}$ (i.e., the sizes of the nodes obtained by merging each e_{i_j}, $j = 1, 2, \ldots, k-1$, have been updated by applying (6.1)) in this order. ■

From the discussion given so far (in particular the assumption on selectivity invariance), it is evident that $C(\pi)$ gives the total size of intermediate relations generated during executing u joins in the order specified by π.

6.3 NP-hardness of OPTJOIN

6.3.1 Main Theorem

This section shows the following main theorem.

Theorem 6.1: OPTJOIN is NP-hard. ■

Since OPTJOIN is an optimization problem, we show this result by proving that the following decision problem version JOIN is NP-hard.

JOIN

Input: a query tree $T = (V, E)$, $V = \{R_1, R_2, \ldots, R_v\}$, $E = \{e_1, e_2, \ldots, e_u\}$ $(u = v - 1)$, the size r_i of each node R_i, the selectivity s_{ij} of each edge $e = (R_i, R_j)$, and a positive number c^*.

Output: yes if there is a permutation π with $C(\pi) \leq c^*$; otherwise no. ■

To prove the NP-hardness [Gar 79] of JOIN, we have to (polynomially) reduce an NP-complete problem B to JOIN, i.e.,

$$B \prec \text{JOIN}.$$

Here, $B \prec$ JOIN means that an arbitrary problem instance Q of B can be transformed into a problem instance P of JOIN in polynomial time in the size of Q, such that P and Q have the same answer. In our discussion, we employ 0-1 KNAPSACK [Gar 79] in the place of B, and show that

$$\text{0-1 KNAPSACK} \prec \text{JOIN}.$$

0-1 KNAPSACK

Input: positive integers $a_0, a_1, \ldots, a_n$ (without loss of generality, we assume $a_0 \geq a_i, i = 1, \ldots, n$).

Output: yes if there exist $x_i \in \{0, 1\}, i = 1, \ldots, n$, such that $a_0 + \sum_{i=1}^{n} a_i x_i = \frac{A}{2} + 1$, where $A = \sum_{i=0}^{n} a_i$; otherwise no. ■

6.3.2 Proof of Theorem 6.1

For a given problem instance of 0-1 KNAPSACK (i.e., positive integers $a_0, a_1, \ldots, a_n$), we consider the problem instance of JOIN defined by the query tree as shown in Figure 6.2, with $(n + 1)$ paths of length 2 and one edge emanating from the center node. Call

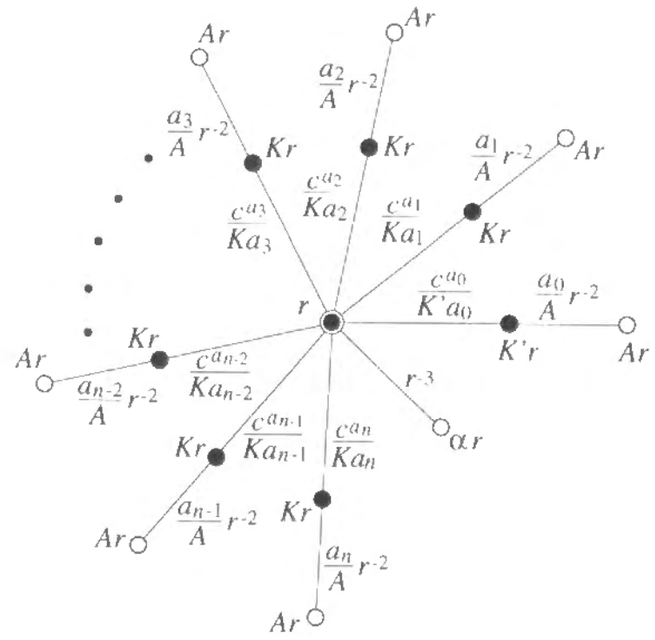


Figure 6.2: A 2-star query tree.

the edges incident to the center node 'inside', and the other edges 'outside'. The weights of nodes and edges are also indicated in the figure, where

$$a_0 > a_i \quad \text{(for all } i\text{)}, \tag{6.3}$$

$$A = \sum_{i=0}^{n} a_i. \tag{6.4}$$

We denote the outside edge with selectivity $\frac{a_i}{Ar^2}$ by e_i and its inside edge by u_i ($i = 0, 1, \ldots, n$). Furthermore, let constants c, r, K, K' and α satisfy

$$c > 3Aa_0 n, \tag{6.5}$$

$$r > 3nc^{3A/2}KA^2, \tag{6.6}$$

$$K > \frac{c^{a_0}}{a_0}, \tag{6.7}$$

$$K' > \frac{3n\alpha c^A}{a_0}, \tag{6.8}$$

$$\alpha = \frac{KAa_0}{2c^{A/2}}. \tag{6.9}$$

Finally, let c^* be given by

$$c^* = \frac{KA}{2} + \frac{7K}{4}. \tag{6.10}$$

Coefficients a_i ($i = 0, 1, \ldots, n$) and c are considered to be exact. However, to keep the input length polynomial in the input length of the original instance of 0-1 KNAPSACK, i.e., n and $\log A$, we assume that $\alpha, c^{a_0}, c^{a_1}, \ldots, c^{a_n}$ are accurate only up to L bits, where L satisfies

$$L > 2 + 2\log nA. \tag{6.11}$$

This condition is necessary for estimating truncation errors in the later discussion. Note that c, r, K and K' can be exact as the conditions on them are given only by inequalities. The transformation from a problem instance of 0-1 KNAPSACK to the above instance of JOIN is obviously done in polynomial time.

Now, the proof proceeds in four stages from 1 to 4 by analyzing the cost of joins in the above instance of JOIN. Stage 1 discusses that one special edge e^* must be joined

first in an optimal join order. After this join, the set of edges is classified in stage 2 into three parts, Σ-, Π- and J-parts, for a given join order. In stage 3, a special edge e_0 and the edges in Π-part are considered, and it is shown that the order of edges in Π-part is irrelevant to the discussion of optimum cost. From the discussion in stages from 1 to 3, it becomes evident that the essential factor in determining an optimal order of joins is how to partition the set of some edges into Π- and Σ-part. It is then shown in stage 4 that computing an optimal partition is equivalent to solving the given instance of 0-1 KNAPSACK, completing the proof of NP-hardness.

Stage 1: The first edge e^* to join

Denote the inside edge with selectivity r^{-3} by e^*. This is introduced to make the initial sizes of all nodes greater than 1, as they represent relational tables. The edge e^* can be the first edge to join among all edges in the graph as shown below. If we join some of other inside edges first, then terms such as

$$\frac{c^{a_0}}{a_0}r^2, \frac{c^{a_1}}{a_1}r^2, \ldots$$

appear in the resulting cost, which are obviously greater than c^* by formulas (6.6) and (6.10). Similarly, if some of outside edges are joined before e^*, we consider a new order by changing e^* to be the first edge to join. This does not change the cost $C(\pi)$ of formula (6.2). Consequently, we can assume without loss of generality that e^* is the first edge to join. Based on this, we consider $C(\pi)$ as the joining cost of all the edges excluding this edge e^*, in the subsequent discussion.

The query graph after having joined e^* is given in Figure 6.3.

Stage 2: Classification of edges into Σ-, Π- and J-part

The next join is executed either on an outside edge or an inside one. But, if we join more than one inside edge consecutively, a term such as

$$\frac{\alpha \cdot c^{a_i} c^{a_j} r}{a_i a_j}$$

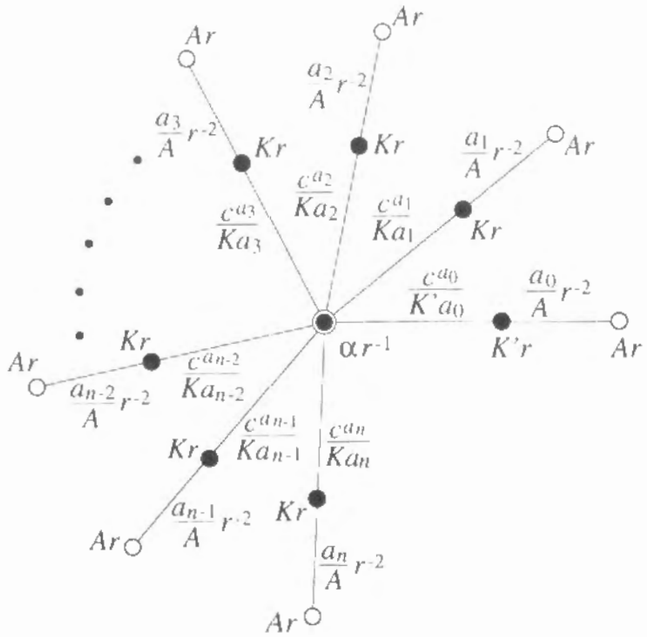


Figure 6.3: Query tree after joining one edge.

appears in the resulting cost, implying that the total cost is greater than c^*. Therefore, we must follow one of the next two rules in order to keep it within c^*.

1. Before joining an inside edge, join the corresponding outside edge adjacent to it, or
2. After joining an inside edge, join the adjacent outside edge immediately.

We call the set of outside edges joined in the manner of (1) as Σ-part, the set of inside and outside edges in (2) as Π-part, and the rest of the inside edges as J-part. These parts are illustrated in Figure 6.4. The joining cost of each part determined by a join order π is denoted by $C_\Sigma(\pi)$, $C_\Pi(\pi)$ or $C_\mathrm{J}(\pi)$, respectively. Therefore, the total cost of formula

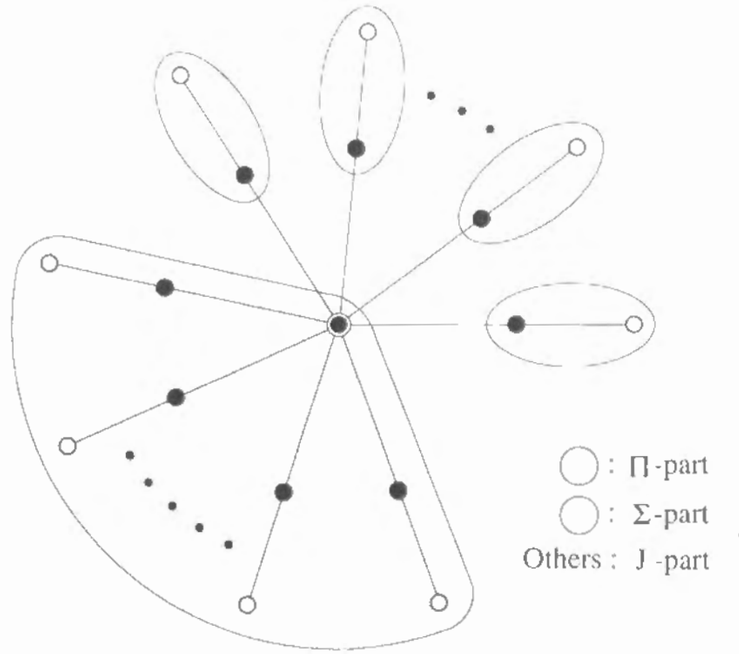


Figure 6.4: Partition of join edges.

(6.2) ignoring the joining cost of e^* is expressed by

$$C(\pi) = C_\Sigma(\pi) + C_\Pi(\pi) + C_\mathrm{J}(\pi) + E, \tag{6.12}$$

where E denotes the truncation error due to the inaccurate representation of coefficients α and c^{a_i} as discussed before formula (6.11).

As to $C_\Sigma(\pi)$, each edge e_i in Σ-part produces such a term as

$$Ka_i. \tag{6.13}$$

This cost is independent of the join order of the edges in Π-part and J-part. Therefore, by considering its definition, we can assume that J-part is executed after Σ-part. But joins in J-part may be interlaced with some joins in Π-part. In any case, $C_\mathrm{J}(\pi)$ never

exceeds 1 as shown below. Whenever two edges e_i and u_i for some i are joined (as an element of Π-part, or J-part after Σ-part), the size of center node (whose size is initially αr^{-1}) is multiplied by

$$Ar \cdot \frac{a_i}{A} r^{-2} \cdot Kr \cdot \frac{c^{a_i}}{Ka_i} = c^{a_i} \quad (> 2).$$

Therefore the size of center node is at most $\alpha \cdot c^{a_0} c^{a_1} \cdots c^{a_n} \cdot r^{-1}$ and we have

$$\begin{aligned} C_{\mathrm{J}}(\pi) &\le \alpha \cdot c^{a_0} c^{a_1} \cdots c^{a_n} \cdot r^{-1} \cdot (n+1) \\ &= \frac{KAa_0}{2c^{A/2}} \cdot c^A \cdot \frac{1}{3nc^{3A/2} KA^2} \cdot (n+1) \quad \text{(by (6.6) and (6.9))} \\ &= \frac{a_0}{6c^A A} \cdot \frac{n+1}{n} < 1. \end{aligned} \tag{6.14}$$

From this, it can be shown that the joins in J-part should be executed after finishing all joins in Π-part. For, if we join an inside edge r_i in J-part among the joins in Π-part, all the intermediate terms in $C_\Pi(\pi)$ (which will be given in formula (6.16)) are multiplied by c^{a_i} after joining u_i. This apparently causes the resulting cost $C_\Pi(\pi)$ to increase at least by 1, if we compare it with the case in which all edges in J-part are joined last. Although $C_{\mathrm{J}}(\pi)$ may decrease by joining J-part first, it always satisfies $C_{\mathrm{J}}(\pi) < 1$ by formula (6.14). Therefore, the cost decrease in J-part cannot compensate the cost increase in Π-part.

Consequently, we have only to consider in the subsequent discussion how to partition the set of all outside edges into Σ- and Π-part (the inside edges adjacent to the edges in Σ-part constitute J-part), and the join order only in Π-part. We say that set $\{a_0, a_1, \ldots, a_n\}$ is partitioned into sets Σ and Π, by denoting $a_i \in \Sigma$ (Π) if and only if an outside edge e_i belongs to Σ-part (Π-part). For this partition, let

$$X = \sum_{a_i \in \Sigma} a_i, \quad Y = \sum_{a_i \in \Pi} a_i \quad \text{(i.e., } X + Y = A\text{)}. \tag{6.15}$$

Stage 3: Special edge e_0 and the join order in Π-part

Let Π-part contain p outside edges and p inside edges, where p satisfies $0 \le p \le n+1$, and let the restriction of a join order π to set $\Pi = \{a_{i_1}, a_{i_2}, \ldots, a_{i_p}\}$ be

$$\pi_\Pi = (u_{i_1}, e_{i_1}, u_{i_2}, e_{i_2}, \ldots, u_{i_p}, e_{i_p}).$$

Then the resulting cost $C_\Pi(\pi)$ becomes

$$\begin{aligned} C_\Pi(\pi) &= \frac{\alpha c^{a_{i_1}}}{a_{i_1}} + \cdots + \frac{\alpha c^{a_{i_1}} c^{a_{i_2}} \cdots c^{a_{i_p}}}{a_{i_p}} \\ &\qquad + (\alpha c^{a_{i_1}} + \cdots + \alpha c^{a_{i_1}} c^{a_{i_2}} \cdots c^{a_{i_p}}) \cdot r^{-1} \qquad (6.16) \\ &= \alpha \left(\frac{c^{a_{i_1}}}{a_{i_1}} + \cdots + \frac{c^{a_{i_1}} c^{a_{i_2}} \cdots c^{a_{i_{p-1}}}}{a_{i_{p-1}}} \right) + \frac{\alpha c^Y}{a_{i_p}} \\ &\qquad + \alpha (c^{a_{i_1}} + \cdots + c^{a_{i_1}} c^{a_{i_2}} \cdots c^{a_{i_p}}) \cdot r^{-1} \qquad (6.17) \\ &< p \cdot \alpha c^Y + p \cdot \alpha c^Y r^{-1} \\ &< (n+1) \cdot \alpha c^Y + 1 \qquad (6.18) \\ &< 3n\alpha c^A, \qquad (6.19) \end{aligned}$$

where the sum of $O(r^{-1})$ terms is less than 1 for the same reason as that of formula (6.14).

Now, call the outside edge with selectivity $\frac{a_0}{A} r^{-2}$ as e_0. If we include e_0 in Σ-part, the cost of joining e_0 is

$$Ar \cdot \frac{a_0}{A} r^{-2} \cdot K'r = K' a_0 \quad (> 3n\alpha c^A), \tag{6.20}$$

according to formula (6.8). However, if edge e_0 is in Π-part, the resulting cost of Π-part is bounded from above by formula (6.19). This means that e_0 must be included in Π-part.

Furthermore, as will be shown in stage 4, $X > \frac{A}{2}$ $\left(Y \le \frac{A}{2} - 1\right)$ is required in order that the total cost does not exceed c^*. In this case of $Y \le \frac{A}{2} - 1$, $C_\Pi(\pi)$ satisfies

$$\begin{aligned} C_\Pi(\pi) &< p \cdot \alpha c^{A/2-1} + p \cdot \alpha c^{A/2-1} r^{-1} \quad \text{(by (6.18))} \\ &< (n+1) \cdot \frac{KAa_0}{2c^{A/2}} \cdot c^{A/2-1} + 1 \quad \text{(by (6.9))} \\ &< 3n \cdot \frac{KAa_0}{2} \cdot \frac{1}{3Aa_0 n} = \frac{K}{2} \quad \text{(by (6.5))}, \end{aligned} \tag{6.21}$$

for any join order π. If this is the case, it will also be shown in stage 4 that $C_\Pi(\pi)$ never affects the decision on partitioning outside edges into Σ- and Π-part. Therefore, the join order of the edges in Π-part is irrelevant to the following discussion. In particular, we

can assume without loss of generality that e_0 is the final edge to join in Π-part (in fact, we can show with some calculation that $C_\Pi(\pi)$ becomes smallest if e_0 is the last edge), and $C_\Pi(\pi)$ of formula (6.17) becomes

$$C_\Pi(\pi) = \frac{\alpha c^Y}{a_0} + D(\pi) = \frac{\alpha c^{(A-X)}}{a_0} + D(\pi), \tag{6.22}$$

where $D(\pi)$ denotes the sum of all terms other than $\frac{\alpha c^{(A-X)}}{a_0}$ in formula (6.17), because $\frac{\alpha c^{(A-X)}}{a_0}$ is obviously dominant.

Stage 4: Partition into Σ- and Π-part

Finally, we consider the partition of $\{a_0, a_1, \ldots, a_n\}$ into Σ- and Π-part. Let us rewrite the total join cost of formula (6.12) determined by a join order π by

$$C(\pi; X) = C_\Sigma(\pi; X) + C_\Pi(\pi; X) + C_\mathrm{J}(\pi) + E. \tag{6.23}$$

In this formula, $C_\Sigma(\pi; X)$ and $C_\Pi(\pi; X)$ is the join cost required in Σ-part and Π-part, respectively, where X is added to indicate their dependence on X of formula (6.15). By formula (6.13), we have

$$C_\Sigma(\pi; X) = KX. \tag{6.24}$$

To evaluate E of formula (6.23), recall that formula (6.11) implies

$$2^L > 4n^2A^2 > 4n^2Aa_0, \tag{6.25}$$

and we have for $Y \leq \frac{A}{2}$ the following upper bound on $E + C_\mathrm{J}(\pi)$:

$$\begin{aligned} |E + C_\mathrm{J}(\pi)\,| &< (n+1) \cdot \alpha c^{A/2} \cdot 2^{-L} + 1 + C_\mathrm{J}(\pi) \\ &\qquad \text{(by (6.18) and the definition of accuracy } L) \\ &< (n+1) \cdot \frac{KAa_0}{2c^{A/2}} \cdot c^{A/2} \cdot \frac{1}{4n^2Aa_0} + 2 \quad \text{(by (6.9) and (6.25))} \\ &< \frac{K}{4n} + 2 < \frac{K}{4} \quad (\text{as } n > 1 \text{ and } K > 16). \end{aligned} \tag{6.26}$$

As noted after formula (6.22), the term $\frac{\alpha c^{(A-X)}}{a_0}$ is dominant in $C_\Pi(\pi; X)$, and $C_\Sigma(\pi; X)$ is linear in X. Furthermore, since α of formula (6.9) is set so that

$$C_\Sigma\left(\pi; \frac{A}{2}\right) \left(= \frac{KA}{2}\right) = \frac{\alpha c^{(A-A/2)}}{a_0} \tag{6.27}$$

holds, $C(\pi; X)$ behaves as illustrated in Figure 6.5. It takes its minimum for any join

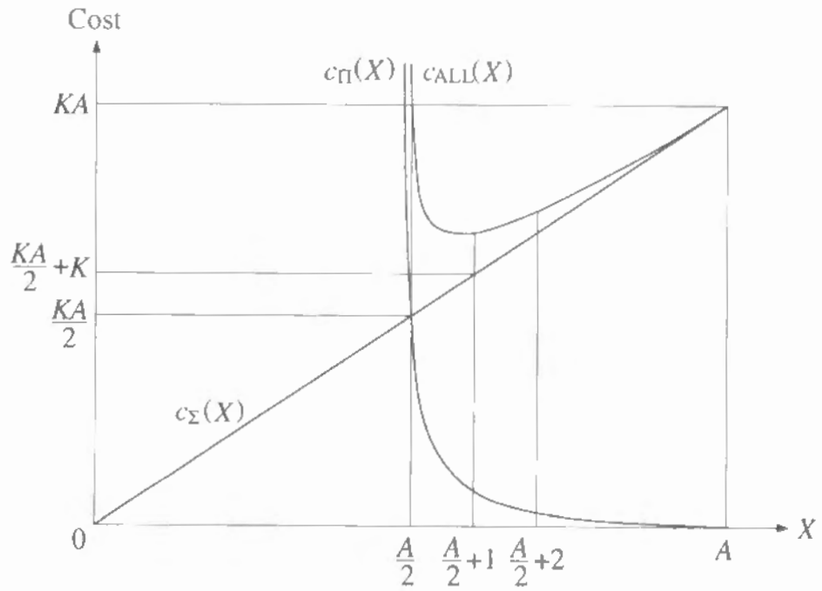


Figure 6.5: Behavior of cost functions.

order π at $X = \frac{A}{2} + 1$ as shown below.

$$\begin{aligned} C\left(\pi; \frac{A}{2}\right) &\geq C_\Sigma\left(\pi; \frac{A}{2}\right) + C_\Pi\left(\pi; \frac{A}{2}\right) + C_\mathrm{J}(\pi) - |E| \\ &> \frac{KA}{2} + \frac{KA}{2} - \frac{K}{4} \quad \text{(by (6.26) and (6.27))} \\ &> \frac{KA}{2} + \frac{7K}{4} \quad (A \geq 4 \text{ is assumed without loss of generality}), \end{aligned} \tag{6.28}$$

$$\begin{aligned} C\left(\pi; \frac{A}{2}+1\right) &\leq C_\Sigma\left(\pi; \frac{A}{2}+1\right) + C_\Pi\left(\pi; \frac{A}{2}+1\right) + C_\mathrm{J}(\pi) + |E| \\ &< K\left(\frac{A}{2}+1\right) + \frac{K}{2} + \frac{K}{4} \quad \text{(by (6.21) and (6.26))} \end{aligned}$$

$$< \frac{KA}{2} + \frac{7K}{4}, \tag{6.29}$$

$$\begin{aligned} C\left(\pi; \frac{A}{2}+2\right) &\geq C_{\Sigma}\left(\pi; \frac{A}{2}+2\right) + C_{\Pi}\left(\pi; \frac{A}{2}+2\right) + C_{\mathrm{J}}(\pi) - |E| \\ &> K\left(\frac{A}{2}+2\right) - \frac{K}{4} \\ &> \frac{KA}{2} + \frac{7K}{4}. \end{aligned} \tag{6.30}$$

Now, recalling $c^* = \frac{KA}{2} + \frac{7K}{4}$ as defined in formula (6.10), we see that $C(\pi; X) \leq c^*$ holds if and only if $X = \frac{A}{2} + 1$ holds. This means that $C(\pi) \leq c^*$ holds for C of formula (6.2), if and only if there exists a partition (Σ, Π) of $\{a_0, a_1, \ldots, a_n\}$ such that $X = \frac{A}{2} + 1$ holds for X of formula (6.15). Then this condition can be realized if and only if the corresponding instance of 0-1 KNAPSACK has a solution. Consequently, 0-1 KNAPSACK is polynomially reduced to JOIN, proving that JOIN and hence OPTJOIN are NP-hard even if query graphs are restricted to be trees. ■

6.4 Restricted Query Trees

We observed that the problem OPTJOIN is NP-hard for arbitrary query trees in the previous section. Even if there is no polynomial time algorithm to compute the optimum join orders for arbitrary query trees, there may exist such algorithms for special query trees. For this purpose, we try to impose some restrictions on the shape of query trees. Some possible restrictions are;

1. restrict the depth of a query tree to be less than k from an appropriate root node,

   or

2. restrict the degree of each node to be less than k.

However, we can state the next corollary immediately.

Corollary 6.2: OPTJOIN with restriction (1) is NP-hard even if $k \leq 2$. ■

This corollary is obvious, since the query trees used in the proof of NP-hardness of OPTJOIN have depth $k \leq 2$. Therefore, restriction (1) does not make OPTJOIN easier.

Moreover, when we consider restriction (2), the following theorem holds.

Theorem 6.3: OPTJOIN with restriction (2) is NP-hard even if $k = 3$. ■

Proof: In order to consider restriction (2) with $k = 3$, let us construct the 2-caterpillar as shown in Figure 6.6. In this case, assume that all the values a_i $(i = 0, 1, \ldots, n)$, A, c,

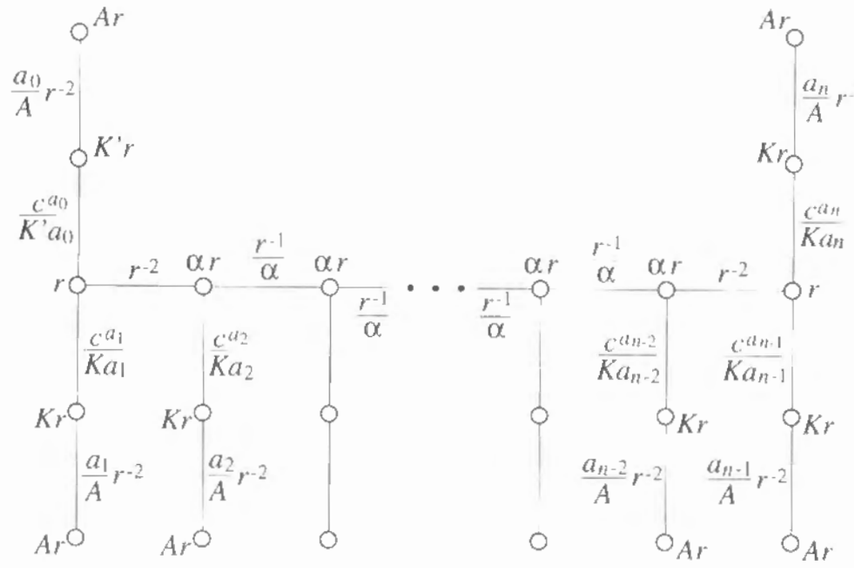


Figure 6.6: A query tree with maximum degree 3.

r, K and K' are equal to those used in Figure 6.2, except that α is given by

$$\alpha = \frac{KAa_0}{4c^{A/2}}.$$

As we can see later, the total cost can be kept within $O(1)$ by using an appropriate order of joins. To attain this, the first edge to be joined is either one of the edges

which is incident to the leaf node with weight Ar, or one of the two outermost edges with selectivity r^{-2} in the trunk, where trunk implies the edges connecting two nodes of degree 3.

In the latter case, the cost c_1 of the edge is

$$c_1 = r^{-2} \cdot r \cdot \alpha r = \alpha.$$

Therefore, if join operations are repeatedly applied to all the edges in the trunk from outside to inside, the total join cost c_2 for these edges is bounded from above by

$$c_2 < n\alpha < \frac{K}{4},$$

as proved in a manner similar to formula (6.21). In this way, we can keep the cost within $O(1)$.

Since this cost c_2 does not give any essential influence to the total cost, as we will see later, we can assume without loss of generality that all the edges connecting nodes of degree 3 are joined first. Then, the query tree resulting after such join operations tuns out to be the same as that of Figure 6.3.

When we consider to execute join operations to the query tree of Figure 6.3, we have to pay attention to the following two values, i.e., the total sum of $O(1)$ terms in the cost of Π-part c_3 in the case of $Y \leq \frac{A}{2} - 1$, and the total errors $E_{\max}$. But these values are bounded as follows.

$$c_3 < \alpha c^{A/2-1} \cdot p < \frac{K}{4},$$

$$E_{\max} < \alpha c^{A/2-1} \cdot 2^{-L} \cdot n + 2 < \frac{K}{8}.$$

So, the total value E of all those errors is bounded from above by

$$E < c_2 + c_3 + 2E_{\max} < \frac{K}{4} + \frac{K}{4} + 2 \cdot \frac{K}{8} < K.$$

The amount of E is at most K, while the total cost for different value of Y differs at least by K. Therefore, we can reduce all the cost functions c_Π, c_Σ and c_{ALL} to those of

Section 6.3. According to Theorem 6.1, this means that finding the optimum join order afterwards is NP-hard. This proves that OPTJOIN to the query tree of Figure 6.6 is also NP-hard. ■

6.5 Query Trees Solvable in Polynomial Time

6.5.1 Shapes of Query Trees

We introduce the definitions of some special query trees used in the latter discussions in this subsection.

Tree: A *tree* is a one-component graph in which there is no cycle. A *k-tree* is a tree which has k nodes.

Chain: A *chain* is a tree in which every node has a degree less than or equal to 2. A *k-chain* is a chain which has k nodes.

Star: A *star* is a tree which has the center node to which chains are connected. A *k-star* is a star for which all the chains emanating from its center are less than or equal to k in length. In particular, a *1-star* stands for a star which has one center node to which all the other nodes are adjacent.

Caterpillar: A *caterpillar* is a tree such that it contains one chain called the *trunk*, to which other chains are connected. A *k-caterpillar* is a caterpillar in which the length of each chain connected to the trunk is less than or equal to k. In other words, a k-caterpillar is considered to be a sequence of k-stars connected by a trunk.

A 2-star and a 2-caterpillar are already seen for proving Theorem 6.1 and Theorem 6.3, respectively, in the previous discussions.

6.5.2 Stars

As a result of previous sections, it became clear that query trees must be severely restricted to have polynomial time algorithms. Here, we give polynomial time algorithms to the following two kinds of query trees.

Consider a star as shown in Figure 6.7. Node weights r_i and edge weights (selectivity)

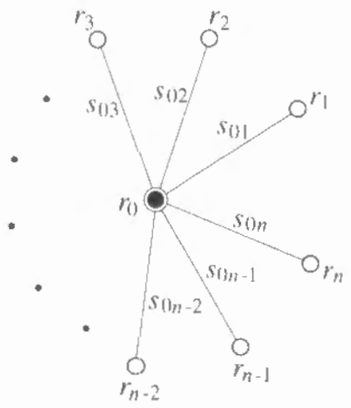


Figure 6.7: A star.

s_{0i} are also given in the figure, where they satisfy $0 < s_{0i} \leq 1$ and $r_i \geq 1$.

Let $M_i = s_{0i}r_i$ $(i = 1, 2, \ldots, m)$, and denote the edge with selectivity s_{0i} by e_i. Furthermore, suppose $M_i \leq M_j$ for $i < j$ without loss of generality. Now consider an arbitrary join order π:

$$\pi = (e_{i_1}, \ldots, e_{i_a}, \ldots, e_{i_b}, \ldots, e_{i_n}).$$

Here, if $i_a > i_b$ holds in π, construct the corresponding join order π' obtained by exchanging e_{i_a} and e_{i_b} in π:

$$\pi' = (e_{i_1}, \ldots, e_{i_b}, \ldots, e_{i_a}, \ldots, e_{i_n}),$$

Then their costs c_π and $c_{\pi'}$ become

$$\begin{aligned} c_\pi = r_0(M_{i_1} + \cdots + M_{i_1} \cdots M_{i_a} + \cdots + M_{i_1} \cdots M_{i_a} \cdots M_{i_b} \\ + \cdots + M_{i_1} \cdots M_{i_a} \cdots M_{i_b} \cdots M_{i_n}), \\ c_{\pi'} = r_0(M_{i_1} + \cdots + M_{i_1} \cdots M_{i_b} + \cdots + M_{i_1} \cdots M_{i_b} \cdots M_{i_a} \\ + \cdots + M_{i_1} \cdots M_{i_b} \cdots M_{i_a} \cdots M_{i_n}) \end{aligned}$$

and obviously satisfy $c_{\pi'} \leq c_\pi$. This shows that joining the edges in nondecreasing order of $M_i = s_{0i}r_i$ is optimum. We show this procedure in Program 6.5. The computational time to find such an optimum order is $O(n \log n)$, which is required for sorting $s_{0i}r_i, i = 1, 2, \ldots, n$.

```
Procedure Star
Input: s(01), ..., s(0n), r(0), ..., r(n)
Output: Optimum join order π
1   begin
2     Sort M(i) = s(0i)r(i) by the increasing order
        into (s(0j_1)r(j_1), ..., s(0j_n)r(j_n));
3     π = (e(j_1), ..., e(j_n));
4   end.
```

Program 6.5: Procedure star.

6.5.3 Chains

Consider a chain as shown in Figure 6.8. In this case, we can apply the following algorithm based on dynamic programming.

First of all, we define H_{ij} by

$$H_{ij} = \prod_{k=i}^{j-1} s_{kk+1} \prod_{k=i}^{j} r_{k}, \quad 0 \leq i < j \leq n.$$

Figure 6.8: A chain.

Each H_{ij} denotes the weight of the node when R_i, R_{i+1}, ..., R_j are joined into one node. Then, denoting the minimum cost of joining all nodes between R_i to $R_j, i < j$, by c_{ij}, the following recursion holds.

$$\begin{cases} c_{ii} = 0, \quad i = 1, 2, \ldots, n, \\ c_{ij} = \min\limits_{0 \le k \le j-i-1} [c_{i,i+k} + c_{i+k+1,j} + H_{ij}], \\ \qquad\qquad \text{for all } i, j \text{ such that } 1 \le i < j \le n. \end{cases} \tag{6.31}$$

The above formulas express that the cost c_{ij} of joining a subchain from R_i to R_j into one node is computed by regarding that the last edge to join is (r_{i+k}, r_{i+k+1}). Since the break point k is not known in advance, minimum is taken over all k. Finally, the optimal total cost c^* for computing joins of a chain of length n is given by

$$c^* = c_{0n}.$$

We summarize these procedures in Program 6.6, where $b(i,j)$ denotes the optimum break point k between the interval (i,j), the $break(i,j)$ denotes the function that derive the optimum break point, and π^R denotes the reverse order of one of the optimum join order π. This procedure in Program 6.6 can be executed by computing c_{ij} in the nondecreasing order of $|j-i|$. Since there are $\frac{n(n-1)}{2}$ possible pairs of i and j, the computational time of this algorithm is $O(n^3)$.


```
Procedure Chain
Input: s(01), s(12), ..., s(n-1n), r(0), ..., r(n)
Output: Optimum join order π and minimum cost c*
 1    begin
 2    for i := 0 to n do begin
 3      c(i,i) := 0; H(i,i) := r(i);
 4      for j := i+1 to n do begin
 5          H(i,j) := H(i,j-1) * s(j-1j) * r(j);
 6        end
 7      end
 8    for i := 0 to n-1 do begin
 9      for j := i+1 to n-1 do begin
10        c(i,j) := c(i,i) + c(i+1,j) + H(i,j);
11        for k := 0 to j-i do begin
12          if c(i,j) > c(i,i+k)+c(i,i+k)+c(i+k+1,j)+H(i,j) then
13            c(i,j) := c(i,i+k)+c(i,i+k)+c(i+k+1,j)+H(i,j);
14            b(i,j) := k;
15          end
16        end
17      end
18    begin
19      break(0,n); π is the reverse order of π^R;
20    end.

function break(i,j)
21    begin
22    while i < j do begin
23      if j := i+1 then begin
24        π^R := append(π^R, e(i,i+1)); break(i,i);
25        end
26      else do begin
27        π^R := append(π^R, e(b(i,j), b(i,j)+1));
28        break(i,b(i,j)); break(b(i,j)+1,j);
29        end
30      end
31    end.
```

Program 6.6: Procedure chain.

6.6 Conclusion

In this chapter, we introduced the problem OPTJOIN that finds the optimum join order that minimizes the sum of intermediate table sizes generated by the merge-scan algorithm, and showed that it is NP-hard for general query trees and that it can only be solved in polynomial time for some special query trees. These results are noteworthy because they exhibit the range, within which the exact optimization of join order can be carried out. The classification is shown in Figure 6.9.

Since 0-1 KNAPSACK, which is known to be weakly NP-hard [Ibarr 75], is used in this paper to reduce it into JOIN, JOIN may be only weakly NP-hard. Therefore, there may exist polynomial time approximation schemes for OPTJOIN, which are similar to those proposed for 0-1 KNAPSACK (e.g., [Ibarr 75]). This is a topic for future research.

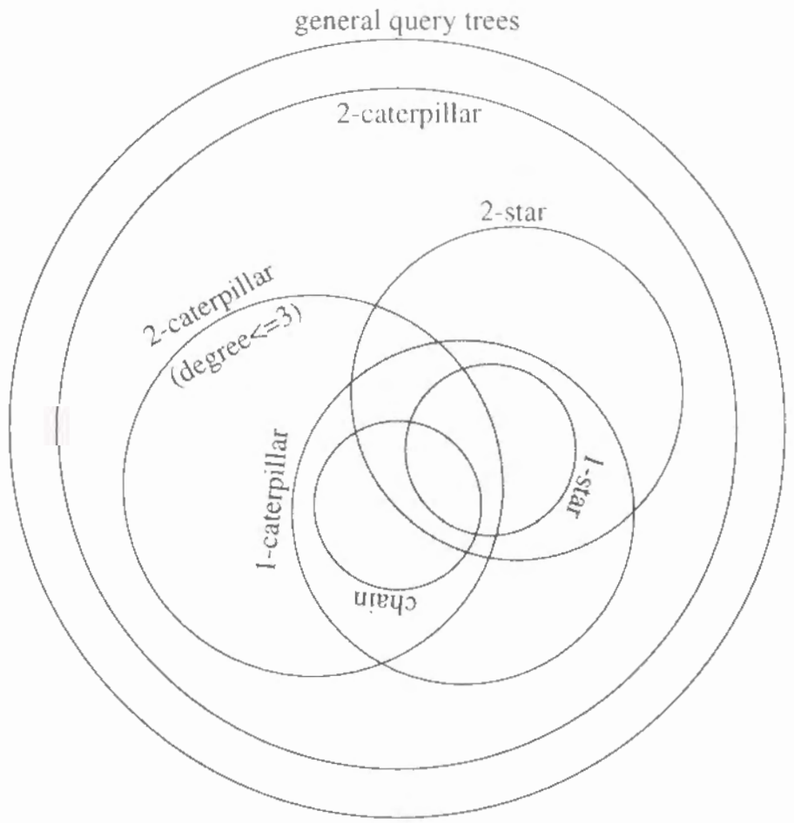


Figure 6.9: Classification of query trees.

Chapter 7

Conclusion

Throughout this thesis, we have developed means to optimize the cost of processing queries for relational or deductive databases.

Our contribution to the field of query optimization in database systems is; we defined the costs of operations in relational algebra in terms of the size of the relation assuming that queries in the relational data model are given in the form of datalog in general, and considering that the procedure of processing such queries is divided into a sequence of operations in relational algebra. Then we estimate the costs by deriving the formulas expressing the sizes of the resulting relations after applying such operations. Among those operations, we paid most of our attention to the operations of join and transitive closure, as they are recognized to be most expensive.

Considering that the processing cost of queries strongly depends on the distributions of elements in the given relations, we proposed a new method that enables us to make cost evaluation for different distributions of data, while very few ideas that take into account the data skewness were seen before.

Our achievements, especially those in Chapter 5, also contribute to the field of graph theory, the reliability of networks, and so on. Since there have been very few work in the

theory of random 'directed' graphs, our results will have the possibility to stimulate the research work in this field.

In this thesis, we have focused on one corner of the field of optimization of query processing that had been paid relatively less attention before. Therefore, we believe that this thesis add some new results to the existing field, and at the same time we hope that our studies will shed light on an unexplored but important field.

Bibliography

[Agr 87] Agrawal, R. and Jagadish, H. V., "Direct algorithms for computing the transitive closure of database relations", *Proceedings of the 13th International Conference on VLDB*, pp. 255–266, 1987.

[Aho 74] Aho, A. V., Hopcroft, J. E. and Ullman, J. D., *The Design of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[Ban 86a] Bancilhon, F. and Ramakrishnan, R., "An amateur's introduction to recursive query processing strategies", *Proceedings of ACM International Conference on Management of Data*, pp. 16–52, 1986.

[Ban 86b] Bancilhon, F., Maier, D., Sagiv, Y. and Ullman, J. D., "Magic sets and other strange ways to implement logic programs", *Proceedings of the 5th ACM SIGACT-SIGMOD Symposium on Principle of Database Systems*, pp. 1–15, 1986.

[Bay 85a] Bayer, R., "Database technology for expert systems", *International GI-Kongress, 85, Wissenbasiete Systeme, Informatik Fachberichte 112*, pp. 1–16, 1985.

[Bay 85b] Bayer, R., "Query evaluation and recursion in deductive database systems", Technical Report TUM-I8503, Technical University Munich, 1985.

[Bee 87] Beeri, C. and Ramakrishnan, R., "On the power of magic", *Proceedings of the 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database*

Systems, pp. 269–283, 1987.

[Bol 82] Bollobás, B., "Long paths in sparse random graphs", *Combinatorica*, 2: 3, pp. 223–228, 1982.

[Bol 85] Bollobás, B., *Random Graphs*, Academic Press Inc. Ltd., London, 1985.

[Cer 87] Ceri, S. and Tanca, L., "Optimization of systems of algebraic equations for evaluating datalog queries", *Proceedings of the 13th International Conference on VLDB*, pp. 31–41, 1987.

[Cha 73] Chang, C. L. and Lee, R. C. T., *Symbolic Logic and Mathematical Theorem Proving*, Academic Press, New York, 1973.

[Cod 70] Codd, E. F, "A relational model of data for large shared data banks", *Communications of ACM*, 13: 6, pp. 377–387, 1970.

[Gard 89] Gardarin, G. and Valduriez, P., *Relational Databases and Knowledge Bases*, Addison-Wesley Publishing Co., Mass., 1989.

[Gar 79] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Fransisco, 1979.

[Gra 93] Graefe, G., "Query evaluation techniques for large databases", *ACM Computing Surveys*, 25: 3, pp. 73–170, 1993.

[Gün 86] Güntzer, U., Kiessling, W. and Bayer, R., "On the evaluation of recursion in (deductive) database systems by efficient differential fixpoint iteration", Technical Report TUM-I8603, Technical University Munich, 1986.

[Gup 92] Gupta, A. and Mumick, I. S., "Magic-sets transformation in nonrecursive systems", *Proceedings of the 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems*, pp. 354–367, 1992.

[Haa 92] Haas, P. J. and Swami, A. N., "Sequential sampling procedures for query size estimation", *Proceedings of the ACM International Conference on Management of Data*, pp. 341–350, 1992.

[Han 93] Han, J., "Compilation and evaluation of linear mutual recursions", *Information Sciences*, 69, pp. 157–183, 1993.

[Hen 84] Henschen, L. J. and Naqvi, S. A., "On compiling queries in recursive first order databases", *J. ACM*, 31: 1, pp. 47–85, 1984.

[Ibara 84] Ibaraki, T. and Kameda, T., "On the optimal nesting order for computing N-relational joins", *ACM TODS*, 9: 3, pp. 482–502, 1984.

[Ibarr 75] Ibarra, O. H. and Kim, C. E., "Fast approximation algorithms for the knapsack and sum of subset problems", *J. ACM*, 22: 4, pp. 463–468, 1975.

[Ioa 86] Ioannidis, Y. E., "On the computation of the transitive closure of relational operators", *Proceedings of the 14th International Conference on VLDB*, pp. 403–411, 1986.

[Ioa 88] Ioannidis, Y. E. and Ramakrishnan, R., "Efficient transitive closure algorithms", *Proceedings of the 14th International Conference on VLDB*, pp. 382–394, 1988.

[Jak 91] Jakobsson, H., "Mixed-approach algorithms for transitive closure", *Proceedings of the 10th ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems*, pp. 199–205, 1991.

[Jak 92] Jakobsson, H., "On tree-based techniques for query evaluation", *Proceedings of the 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems*, pp. 380–392, 1992.

[Kif 86] Kifer, M. and Lozinskii, E. L., "Filtering dataflow in deductive databases", *Proceedings of the 1st International Conference on Database Theory*, pp. 186–202, *Lecture Notes in Computer Science*, Springer Verlag, Berlin, 1986.

[Kim 80] Kim, W., "A new way to compute the product and join of relations", *Proceedings of the ACM International Conference on Management of Data*, pp. 179–187, 1980.

[Knu 73] Knuth, D. E., *The Art of Computer Programmings, Vol.3: Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.

[Knu 75] Knuth, D. E., *The Art of Computer Programmings, Vol.2: Seminumerical Algorithms*, Addison-Wesley, Reading, Mass., 1975.

[Kri 86] Krishnamurthy, R., Boral, H. and Zaniolo, C., "Optimization of nonrecursive queries", *Proceedings of the 12th International Conference on VLDB*, pp. 128–137, 1986.

[Lip 89] Lipton, R. J. and Naughton, J. F., "Estimating the size of generalized transitive closures", *Proceedings of the 15th International Conference on VLDB*, pp. 165–171, 1989.

[Lip 90] Lipton, R. J. and Naughton, J. F., "Query size estimation by adaptive sampling", *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems*, pp. 40–46, 1990.

[Lu 87] Lu, H., "New strategies for computing the transitive closures of a database relation", *Proceedings of the 13th International Conference on VLDB*, pp. 267–274, 1987.

[Lyn 88] Lynch, C. A., "Selectivity estimation and query optimization in large databases with highly skewed distributions of column values", *Proceedings of the 14th International Conference on VLDB*, pp. 240–251, 1988.

[McG 59] McGee, W. C., "Generalization: key to successful electronic data processing", *J. ACM*, 6: 1, pp. 1–23, 1959.

[McK 81] McKay, D. P. and Shapiro, S. C., "Using active connection graphs for reasoning with recursive rules", *Proceedings of the 7th IJCAI Conference*, pp.368–374, 1981.

[Miy 89] Miyazaki, N., "Trends in query optimization methods for deductive databases", *Proceedings of Advanced Database System Symposium*, pp.368 374, 1989.

[Pit 83] Pittel, B., "On distribution related to transitive closure of the random finite mappings", *Annals of Probabilistics*, 11, pp. 428–441, 1983.

[Sek 89] Seki, H., "On the power of Alexander templates", *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems*, pp. 150 159, 1989.

[Sip 88] Sippu, S. and Soisalon-Soininen, E., "A generalized transitive closure for relational queries", *Proceedings of the 7th ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems*, pp. 325–332, 1988.

[Ull 85a] Ullman, J. D., *Principles of Database Systems, 2nd Edition*, Computer Science Press, 1982.

[Ull 85b] Ullman, J. D., "Implementation of logical query languages for databases", *ACM TODS*, 10: 3, pp. 289 321, 1985.

[Ull 89a] Ullman, J. D., "Bottom-up beats top-down for datalog", *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems*, pp. 140–149, 1989.

[Ull 89b] Ullman, J. D., *Principles of Database and Knowledge-Base Systems, Vol. 1*, Computer Science Press, 1989.

[Ull 90] Ullman, J. D., *Principles of Database and Knowledge-Base Systems, Vol. 2*, Computer Science Press, 1990.

[Uno 91] Uno, Y. and Ibaraki, T., "Complexity of the optimal join order problem in relational databases", *IEICE Transactions*, E-74: 7, pp. 2067–2075, 1991.

[Uno 92] Uno, Y. and Ibaraki, T., "Approximate evaluation of the computing cost for answering queries in deductive databases" (in Japanese), *IEICE Transactions*, J75-D-I: 9, pp. 855–863, 1992. (English translation in *Systems and Computers in Japan*, 24: 11, pp. 1–12, 1993.)

[Uno 94a] Uno, Y. and Ibaraki, T., "Approximate evaluation to the size of transitive closures of relations" (in Japanese), *IPSJ Transactions*, 35: 7, pp. 1493–1500, 1994.

[Uno 94b] Uno, Y. and Ibaraki, T., "The expected size of transitive closure of a random digraph", (to be submitted).

[War 62] Warshall, S., "A theorem on Boolean matrices", *J. ACM*, 10: 1, pp. 11–12, 1962.

[Wol 90] Wolf, J. L., Dias, D. M. and Yu, P. S., "An effective algorithm for parallelizing sort merge joins in the presence of data skew", *Proceedings of the 2nd International Symposium on Databases in Parallel and Distributed Systems*, pp. 103–115, 1990.

[Wri 73] Wright, E. M., "For how many edges is a digraph almost certainly Hamiltonian?", *Proceedings of the American Mathematical Society*, 41: 2, pp. 384–388, 1973.

[Wri 78] Wright, E. M., "Asymptotic formulas for the number of oriented graphs", *J. Comninatorial Theory*, B: 24, pp. 370-373, 1978.

[You 92] Youn, C., Kim, H. -J., Henschen, L. J. and Han, J., "Classification and compilation of linear recursive queries in deductive databases", *IEEE Transactions on Knowledge and Data Engineering*, 4: 1, pp. 52–67, 1992.

[Zan 86] Zaniolo, C., "Safety and compilation of non-recursive Horn clauses", *Proceedings of the 1st International Conference on Expert Database Systems*, pp. 167–178, 1986.